Relational Representation Learning for Dynamic (Knowledge) Graphs: A Survey

Seyed Mehran Kazemi, Rishab Goel, Kshitij Jain, Ivan Kobyzev,
Akshay Sethi, Peter Forsyth, Pascal Poupart
Borealis AI
{mehran.kazemi,rishab.goel,kshitij.jain,ivan.kobyzev,
akshay.sethi,peter.forsyth,pascal.poupart}@borealisai.com

Abstract

Graphs arise naturally in many real-world applications including social networks, recommender systems, ontologies, biology, and computational finance. Traditionally, machine learning models for graphs have been mostly designed for static graphs. However, many applications involve evolving graphs. This introduces important challenges for learning and inference since nodes, attributes, and edges change over time. In this survey, we review the recent advances in representation learning for dynamic graphs, including dynamic knowledge graphs. We describe existing models from an encoder-decoder perspective, categorize these encoders and decoders based on the techniques they employ, and analyze the approaches in each category. We also review several prominent applications and widely used datasets, and highlight directions for future research.

1 Introduction

In the era of big data, a challenge is to leverage data as effectively as possible to extract patterns, make predictions, and more generally unlock value. In many situations, the data does not consist only of vectors of features, but also relations that form graphs among entities. Graphs naturally arise in social networks (users with friendship relations, emails, text messages), recommender systems (users and products with transactions and rating relations), ontologies (concepts with relations), computational biology (protein-protein interactions), computational finance (web of companies with competitor, customer, subsidiary relations, supply chain graph, graph of customer-merchant transactions), etc. While it is often possible to ignore relations and use traditional machine learning techniques based on vectors of features, relations provide additional valuable information that permits inference among nodes. Hence, graph-based techniques have emerged as leading approaches in the industry for application domains with relational information.

Traditionally, research has been done mostly on static graphs where nodes and edges are fixed and do not change over time. Many applications, however, involve dynamic graphs. For instance, in social media, communication events such as emails and text messages are streaming while friendship relations evolve over time. In recommender systems, new products, new users and new ratings appear every day. In computational finance, transactions are streaming and supply chain relations are continuously evolving. As a result, the last few years have seen a surge of works on dynamic graphs. This survey focuses precisely on dynamic graphs. Note that there are already many good surveys on static graphs [88, 251, 27, 48, 172, 227]. There are also several surveys on techniques for dynamic graphs [15, 254, 208, 2, 6], but they do not review recent advances in neural representation learning.

We present a survey that focuses on recent representation learning techniques for dynamic graphs. More precisely, we focus on reviewing techniques that either produce time-dependent embeddings that capture the essence of the nodes and edges of evolving graphs or use embeddings to answer various questions such as node classification, event prediction/interpolation, and link prediction. Accordingly, we use an encoder-decoder framework to categorize and analyze techniques that encode various aspects of graphs into embeddings and other techniques that decode embeddings into predictions. We survey techniques that deal with discrete- and/or continuous-time events.

The survey is structured as follows. Section 2 introduces the notation and provides some background about static/dynamic graphs, inference tasks, and learning techniques. Section 3 provides an overview of representation learning techniques for static graphs. This section is not meant to be a survey, but rather to introduce important concepts that will be extended for dynamic graphs. Section 4 categorizes decoders for dynamic graphs into time-predicting and time-conditioned decoders, and surveys the decoders in each category. Section 5 describes encoding techniques that aggregate temporal observations and static features, use time as a regularizer, perform decompositions, traverse dynamic networks with random walks, and model observation sequences with various types of processes (e.g., recurrent neural networks). Section 6 describes briefly other lines of work that do not conform to the encoder-decoder framework such as statistical relational learning, and topics related to dynamic (knowledge) graphs such as spatiotemporal graphs and the construction of dynamic knowledge graphs from text. Section 7 reviews important applications of dynamic graphs with representative tasks. A list of static and temporal datasets is also provided with a brief summary of their properties. Section 8 concludes the survey with a discussion of several open problems and possible research directions.

2 Background and Notation

In this section, we define our notation and provide the necessary background for readers to follow the rest of the survey. A summary of the main notation and abbreviations can be found in Table 1.

We use lower-case letters to denote scalars, bold lower-case letters to denote vectors, and bold upper-case letters to denote matrices. For a vector $\mathbf{z}$, we represent the i^{th} element of the vector as $\mathbf{z}[i]$. For a matrix $\mathbf{A}$, we represent the i^{th} row of $\mathbf{A}$ as $\mathbf{A}[i]$, and the element at the i^{th} row and j^{th} column as $\mathbf{A}[i][j]$. $||\mathbf{z}||_i$ represents norm i of a vector $\mathbf{z}$ and $||\mathbf{Z}||_F$ represents the Frobenius norm of a matrix $\mathbf{Z}$. For two vectors $\mathbf{z_1} \in \mathbb{R}^{d_1}$ and $\mathbf{z_2} \in \mathbb{R}^{d_2}$, we use $[\mathbf{z}_1; \mathbf{z}_2] \in \mathbb{R}^{d_1+d_2}$ to represent the concatenation of the two vectors. When $d_1 = d_2 = d$, we use $[\mathbf{z}_1 \ \mathbf{z}_2] \in \mathbb{R}^{d\times 2}$ to represent a $d \times 2$ matrix whose two columns correspond to $\mathbf{z}_1$ and $\mathbf{z}_2$ respectively. We use $\odot$ to represent element-wise (Hadamard) multiplication. We represent by $\mathbf{I_d}$ the identity matrix of size $d \times d$. $\texttt{vec}(\mathbf{A})$ vectorizes $\mathbf{A} \in \mathbb{R}^{d_1\times d_2}$ into a vector of size $d_1 d_2$. $\texttt{diag}(\mathbf{z})$ turns $\mathbf{z} \in \mathbb{R}^d$ into a diagonal matrix $\mathbf{A} \in \mathbb{R}^{d\times d}$ that has the values of $\mathbf{z}$ on its main diagonal. We denote the transpose of a matrix $\mathbf{A}$ as $\mathbf{A}'$.

2.1 Static Graphs

A *(static) graph* is represented as $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ where $\mathcal{V} = \{\mathsf{v}_1, \mathsf{v}_1, \dots, \mathsf{v}_N\}$ is the set of vertices and $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ is the set of edges. Vertices are also called *nodes* and we use the two terms interchangeably. Edges are also called *links* and we use the two terms interchangeably.

Several matrices can be associated with a graph. An *adjacency matrix* $\mathbf{A} \in \mathbb{R}^{N\times N}$ is a matrix where $\mathbf{A}[i][j] = 0$ if $(\mathsf{v_i}, \mathsf{v_j}) \notin \mathcal{E}$; otherwise $\mathbf{A}[i][j] \in \mathbb{R}_+$ represents the weight of the edge. For unweighted graphs, all non-zero $\mathbf{A}[i][j]$s are 1. A *degree matrix* $\mathbf{D} \in \mathbb{R}^{N\times N}$ is a diagonal matrix where $\mathbf{D}[i][i] = \sum_{j=1}^{N} \mathbf{A}[i][j]$ represents the degree of v_i. A *graph Laplacian* is defined as $\mathbf{L} = \mathbf{D} - \mathbf{A}$.

A graph is *undirected* if the order of the nodes in the edges is *not* important. For an undirected graph, the adjacency matrix is symmetric, i.e. $\mathbf{A}[i][j] = \mathbf{A}[j][i]$ for all i and j. A graph is *directed* if the order of the nodes in the edges is important. Directed graphs are also called *digraphs*. For an edge $(\mathsf{v}_i, \mathsf{v}_j)$ in a digraph, we call v_i the *source* and v_j the *target* of the edge. A graph is *bipartite* if the nodes can be split into two groups where there is no edge between any pair of nodes in the same group. A *multigraph* is a graph where multiple edges can exist between two nodes. A graph is *attributed* if each node is associated with a number of properties representing its characteristics. For a node v in an attributed graph, we let $\mathbf{x}_\mathsf{v}$ represent the attribute values of v. When all nodes have the same attributes, we represent all attribute values of the nodes by a matrix $\mathbf{X}$ whose i^{th} row corresponds to the attribute values of v_i.

A *knowledge graph (KG)* is a multi-digraph with labeled edges [118], where the label represents the type of the relationship. Let $\mathcal{R} = \{\mathsf{r}_1, \mathsf{r}_2, \dots, \mathsf{r}_\mathsf{M}\}$ be a set of relation types. Then $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{R} \times \mathcal{V}$. A

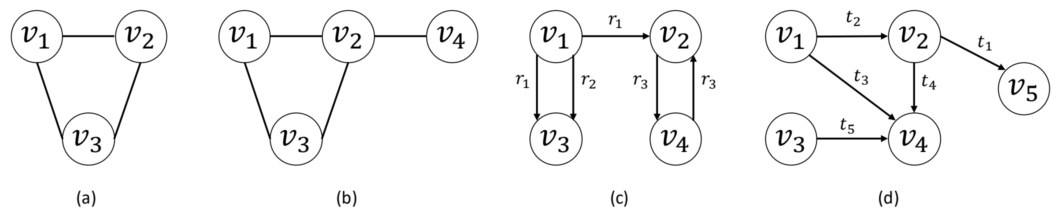

Figure 1: Four graphs to be used as running examples throughout the survey. (a) and (b) are two examples of undirected graphs. They can be also considered as two snapshots of a discrete-time dynamic graph. (c) is an example of a knowledge graph. (d) is an example of a continuous-time dynamic graph where the only possible event/observation is edge addition.

KG can be attributed in which case each node $\mathsf{v} \in \mathcal{V}$ is associated with a vector $\mathbf{x}_\mathsf{v}$ of attribute values. A digraph is a special case of a KG with only one relation. An undirected graph is a special case of a KG with only one symmetric relation.

Example 1. Figure 1(a) represents an undirected graph with three nodes v_1, v_2 and v_3 and three edges $(\mathsf{v}_1, \mathsf{v}_2)$, $(\mathsf{v}_1, \mathsf{v}_3)$ and $(\mathsf{v}_2, \mathsf{v}_3)$. Figure 1(b) represents a graph with four nodes and four edges. The adjacency, degree, and Laplacian matrices for the graph in Figure 1(b) are as follows:

$$\mathbf{A} = \begin{bmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \quad \mathbf{D} = \begin{bmatrix} 2 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \mathbf{L} = \begin{bmatrix} 2 & -1 & -1 & 0 \\ -1 & 3 & -1 & -1 \\ -1 & -1 & 2 & 0 \\ 0 & -1 & 0 & 1 \end{bmatrix}$$

where the i^{th} row (and the i^{th} column) corresponds to v_i. Since the graph is undirected, $\mathbf{A}$ is symmetric. Figure 1(c) represents a KG with four nodes v_1, v_2, v_3 and v_4, three relation types r_1, r_2, and r_3, and five labeled edges as follows:

$$(\mathsf{v}_1, \mathsf{r}_1, \mathsf{v}_2) \quad (\mathsf{v}_1, \mathsf{r}_1, \mathsf{v}_3) \quad (\mathsf{v}_1, \mathsf{r}_2, \mathsf{v}_3) \quad (\mathsf{v}_2, \mathsf{r}_3, \mathsf{v}_4) \quad (\mathsf{v}_4, \mathsf{r}_3, \mathsf{v}_2)$$

The KG in Figure 1(c) is directed and is a multigraph as there are, e.g., two edges (with the same direction) between v_1 and v_3.

2.2 Dynamic Graphs

We represent a *continuous-time dynamic graph (CTDG)* as $\mathcal{CTDG} = (\mathcal{G}, \mathcal{O})$ where $\mathcal{G}$ is a static graph representing an initial state of a dynamic graph at time t_0 and $\mathcal{O}$ is a set of observations/events where each observation is a tuple $(event\ type, event, timestamp)$. An event type can be an edge addition, edge deletion, node addition, node deletion, node splitting, node merging, etc. At any point $t \geq t_0$ in time, a snapshot $\mathcal{G}^t$ (corresponding to a static graph) can be obtained from $\mathcal{CTDG}$ by updating $\mathcal{G}$ sequentially according to the observations $\mathcal{O}$ that occurred before (or at) time t (sometimes, the update may require aggregation to handle multiple edges between two nodes).

A *discrete-time dynamic graph (DTDG)* is a sequence of snapshots from a dynamic graph sampled at regularly-spaced times. Formally, $\mathcal{DTDG} = \{\mathcal{G}^1, \mathcal{G}^2, \ldots, \mathcal{G}^T\}$ where $\mathcal{G}^t = \{\mathcal{V}^t, \mathcal{E}^t\}$. We use the term *dynamic graph* to refer to both DTDGs and CTDGs. Compared to a CTDG, a DTDG may lose information by looking only at some snapshots of the graph over time, but developing models for DTDGs may be generally easier. In particular, a model developed for CTDGs may be used for DTDGs, but the reverse is not necessarily true.

An *undirected dynamic graph* is a dynamic graph where at any time t, $\mathcal{G}^t$ is an undirected graph. A *directed dynamic graph* is a dynamic graph where at any time t, $\mathcal{G}^t$ is a digraph. A *bipartite dynamic graph* is a dynamic graph where at any time t, $\mathcal{G}^t$ is a bipartite graph. A *dynamic KG* is a dynamic graph where at any time t, $\mathcal{G}^t$ is a KG.

Example 2. Let $\mathcal{CTDG} = (\mathcal{G}, \mathcal{O})$ be a CTDG where $\mathcal{G}$ is a graph with five nodes v_1, v_2, v_3, v_4 and v_5 and with no edges between any pairs of nodes, and $\mathcal{O}$ is:

$$\{(AddEdge, (\mathsf{v}_2, \mathsf{v}_5), t_1), (AddEdge, (\mathsf{v}_1, \mathsf{v}_2), t_2), (AddEdge, (\mathsf{v}_1, \mathsf{v}_4), t_3), (AddEdge, (\mathsf{v}_2, \mathsf{v}_4), t_4), (AddEdge, (\mathsf{v}_3, \mathsf{v}_4), t_5)\}$$

Symbol(s) or abbreviation	Meaning
DTDG, CTDG	Discrete-Time and Continuous-Time Dynamic Graph
KG	Knowledge Graph
$\mathcal{G}, \mathcal{V}, \mathcal{E}$	Graph, nodes, and edges.
N	Number of nodes in a graph
$\mathbf{A}, \mathbf{L}, \mathbf{D}, \mathbf{X}$	Adjacency, Laplacian, degree, and attribute matrices of a graph
$\mathcal{O}$	Set of observations for a CTDG
$\mathbf{W}$	Matrix of learnable weights
$\mathcal{G}^t, \mathcal{V}^t, \mathcal{E}^t, \mathbf{A}^t$	Graph, nodes, edges, and adjacency matrix at time t.
v, u	Two generic nodes in a graph.
T	The number of snapshots in a DTDG
`EMB`	The embedding function
$[\mathbf{z}_1; \mathbf{z}_2]$	Concatenation of two vectors $\mathbf{z}_1$ and $\mathbf{z}_2$
ϕ, σ	A generic and the Sigmoid activation function
`vec(.)`	Vectorized view of the input matrix or tensor
$\lvert\lvert\mathbf{z}\rvert\rvert_i, \lvert\lvert\mathbf{Z}\rvert\rvert_F$	Norm i of $\mathbf{z}$, and Frobenius norm of $\mathbf{Z}$.
$\mathbf{A}', \mathbf{z}'$	Transpose of a matrix and a vector

Table 1: Summary of the main notation and abbreviations.

$\mathcal{CTDG}$ may be represented graphically as in Figure 1(d). The only type of observation in this dynamic graph is the addition of new edges. The second element of each observation corresponding to an edge addition represents the source and the target nodes of the new edge. The third element of each observation represents the timestamp at which the observation was made.

Example 3. Consider an undirected CTDG whose initial state is as in Figure 1(a). Suppose $\mathcal{O}$ is:

$$\{(AddNode, \mathsf{v}_4,\ t_1), (AddEdge, (\mathsf{v}_2, \mathsf{v}_4),\ t_2)\}$$

where $t_1 \leq t_2$. Now consider a DTDG that takes two snapshots from this CTDG, one snapshot at time t_0 and one snapshot at time t_2. The two snapshots of this DTDG look like the graphs in Figure 1(a) and Figure 1(b) respectively.

2.3 Prediction problems

In this survey, we mainly study three general problems for dynamic graphs: *node classification*, *edge prediction*, and *graph classification*. Node classification is the problem of classifying each node into one class from a set of predefined classes. Link prediction is the problem of predicting new links between the nodes. Graph classification is the problem of classifying a whole graph into one class from a set of predefined classes. A high-level description of some other prediction problems can be found in Section 7.1.

Node classification and link prediction can be deployed under two settings: *interpolation* and *extrapolation*. Consider a dynamic graph that has incomplete information from the time interval $[t_0, t_T]$. The *interpolation* problem is to make predictions at some time t such that $t_0 \leq t \leq t_T$. The interpolation problem is also known as the *completion* problem and is mainly used for completing (dynamic) KGs [111, 133, 77, 54]. The *extrapolation* problem is to make predictions at time t such that $t \geq t_T$, i.e., predicting future based on the past. Extrapolation is usually a more challenging problem than the interpolation problem.

Streaming scenario: In the streaming scenario, new observations are being streamed to the model at a fast rate and the model needs to update itself based on these observations in real-time so it can make informed predictions immediately after each observation arrives. For this scenario, a model may not have enough time to retrain completely or in part when new observations arrive. Streaming scenarios are often best handled by CTDGs and often give rise to extrapolation problems.

2.4 The Encoder-Decoder Framework

Following Hamilton *et al.* [88], to deal with the large notational and methodological diversity of the existing approaches and to put the various methods on an equal notational and conceptual footing, we develop an encoder-decoder framework for dynamic graphs. Before describing the encoder-de coder framework, we define a main component in this architecture known as *embedding*.

Definition 1. An *embedding* is a function that maps every node $\mathsf{v} \in \mathcal{V}$ of a graph, and every relation type $\mathsf{r} \in \mathcal{R}$ in case of a KG, to a hidden representation where the hidden representation is typically a tuple of one or more scalars, vectors, and/or matrices of numbers. The vectors and matrices in the tuple are supposed to contain the necessary information about the nodes and relations to enable making predictions about them.

For each node v and relation r, we refer to the hidden representation of v and r as the embedding of v and the embedding of r respectively. When the main goal is link prediction, me works define the embedding function as mapping each pair of nodes into a hidden representation. In these cases, we refer to the hidden representation of a pair (v, u) of nodes as the embedding of the pair (v, u).

Having the above definition, we can now formally define an encoder and a decoder.

Definition 2. An *encoder* takes as input a dynamic graph and outputs an embedding function that maps nodes, and relations in case of a KG, to hidden representations.

Definition 3. A *decoder* takes as input an embedding function and makes predictions (such as node classification, edge prediction, etc.) based on the embedding function.

In many cases (e.g., [123, 87, 241, 17, 173, 63]), the embedding function $\mathtt{EMB}(.)$ maps each node, and each relation in the case of a KG, to a tuple containing a single vector; that is $\mathtt{EMB}(\mathsf{v}) = (\mathbf{z}_\mathsf{v})$ where $\mathbf{z}_\mathsf{v} \in \mathbb{R}^{d_1}$ and $\mathtt{EMB}(\mathsf{r}) = (\mathbf{z}_\mathsf{r})$ where $\mathbf{z}_\mathsf{r} \in \mathbb{R}^{d_2}$. Other works consider different representations. For instance, Kazemi and Poole [115] define $\mathtt{EMB}(\mathsf{v}) = (\mathbf{z}_\mathsf{v}, \overline{\mathbf{z}}_\mathsf{v})$ and $\mathtt{EMB}(\mathsf{r}) = (\mathbf{z}_\mathsf{r}, \overline{\mathbf{z}}_\mathsf{r})$, i.e. mapping each node and each relation to two vectors where each vector has a different usage. Nguyen *et al.* [168] define $\mathtt{EMB}(\mathsf{v}) = (\mathbf{z}_\mathsf{v})$ and $\mathtt{EMB}(\mathsf{r}) = (\mathbf{z}_\mathsf{r}, \mathbf{P}_\mathsf{r}, \mathbf{Q}_\mathsf{r})$, i.e. mapping each node to a single vector but mapping each relation to a vector and two matrices. We will describe these approaches (and many others) in the upcoming sections.

A *model* corresponds to an encoder-decoder pair. One of the benefits of describing models in an encoder-decoder framework is that it allows for creating new models by combining the encoder from one model with the decoder from another model when the hidden representations produced by the encoder conform to the hidden representations consumed by the decoder.

2.4.1 Training

For many choices of an encoder-decoder pair, it is possible to train the two components end-to-end. In such cases, the parameters of the encoder and the decoder are typically initialized randomly. Then, until some criterion is met, several epochs of stochastic gradient descent are performed where in each epoch, the embedding function is produced by the encoder, predictions are made based on the embedding function by the decoder, the error in predictions is computed with respect to a loss function, and the parameters of the model are updated based on the loss.

For node classification and graph classification, the loss function can be any classification loss (e.g., cross entropy loss). For link prediction, typically one only has access to positive examples corresponding to the links already in the graph. A common approach in such cases is to generate a set of negative samples where negative samples correspond to edges that are believed to have a low probability of being in the graph. Then, having a set of positive and a set of negative samples, the training of a link predictor turns into a classification problem and any classification loss can be used. The choice of the loss function depends on the application.

2.5 Expressivity

The expressivity of the models for (dynamic) graphs can be thought of as the diversity of the graphs they can represent. Depending on the problem at hand (e.g., node classification, link prediction, graph classification, etc.), the expressivity can be defined differently. We first provide some intuition on the importance of expressivity using the following example.

Example 4. Consider a simple encoder for a KG that maps every node to a tuple containing a single scalar representing the number of incoming edges to the node (regardless of the labels of the edges). For the KG in Figure 1(c), this encoder will output an embedding function as:

$$\mathtt{EMB}(\mathsf{v}_1) = (0) \qquad \mathtt{EMB}(\mathsf{v}_2) = (2) \qquad \mathtt{EMB}(\mathsf{v}_3) = (2) \qquad \mathtt{EMB}(\mathsf{v}_4) = (1)$$

No matter what decoder we use, since $\texttt{EMB}(\mathsf{v}_2)$ and $\texttt{EMB}(\mathsf{v}_3)$ are identical, the two nodes will be assigned the same class. Therefore, this model is not expressive enough to represent ground truths where v_2 and v_3 belong to different classes.

From Example 4, we can see why the expressivity of a model may be important. In this regard, one may favor models that are fully expressive, where we define full expressivity for node classification as follows (a model in the following definitions corresponds to an encoder-decoder pair):

Definition 4. A model $\mathcal{M}$ with parameters Θ is *fully expressive with respect to node classification* if given any graph $\mathcal{G}$ and any ground truth Ω of class assignments for all nodes in the graph, there exists an instantiation of Θ that classifies the nodes of $\mathcal{G}$ according to Ω.

A similar definition can be given for full expressivity of a model with respect to link prediction and graph classification.

Definition 5. A model $\mathcal{M}$ with parameters Θ is *fully expressive with respect to link prediction* if given any graph $\mathcal{G}$ and any ground truth Ω indicating the existence or non-existence of a (labeled) edge for all node-pairs in the graph, there exists an instantiation of Θ that classifies the node-pairs of $\mathcal{G}$ according to Ω.

Definition 6. A model $\mathcal{M}$ with parameters Θ is *fully expressive with respect to graph classification* if given any set $\{\mathcal{G}_1, \mathcal{G}_2, \dots, \mathcal{G}_n\}$ of non-isomorphic graphs and any ground truth Ω of class assignments for all graphs in the set, there exists an instantiation of Θ that classifies the graphs according to Ω.

2.6 Sequence Models

In dynamic environments, data often consists of sequences of observations of varying length. There is a long history of models to handle sequential data without any fixed length. This includes auto-regressive models [5] that predict the next observations based on a window of past observations. Alternatively, since it is not always clear how long the window of part observations should be, hidden Markov models [188], Kalman filters [232], dynamic Bayesian networks [165] and dynamic conditional random fields [212] use hidden states to capture relevant information that might be arbitrarily far in the past. Today, those models can be seen as special cases of recurrent neural networks, which allow rich and complex hidden dynamics.

Recurrent neural networks (RNNs) [70, 44] have achieved impressive results on a range of sequence modeling problems such as language modeling and speech recognition. The core principle of the RNN is that its input is a function of the current data point as well as the history of the previous inputs. A simple RNN model can be formulated as follows:

$$\mathbf{h}^t = \phi(\mathbf{W}_i \mathbf{x}^t + \mathbf{W}_h \mathbf{h}^{t-1} + \mathbf{b}_i) \tag{1}$$

where $\mathbf{x}^t \in \mathbb{R}^{d'}$ is the input at position t in the sequence, $\mathbf{h}^{t-1} \in \mathbb{R}^d$ is a hidden representation containing information about the sequence of inputs until time $t-1$, $\mathbf{W}_i \in \mathbb{R}^{d \times d'}$ and $\mathbf{W}_h \in \mathbb{R}^{d \times d}$ are weight matrices, $\mathbf{b}_i \in \mathbb{R}^d$ represents the vector of biases, ϕ is an activation function, and $\mathbf{h}^t \in \mathbb{R}^d$ is an updated hidden representation containing information about the sequence of inputs until time t. With some abuse of notation, we use $\mathbf{h}^t = \texttt{RNN}(\mathbf{h}^{t-1}, \mathbf{x}^t)$ to represent the output of an RNN operation on a previous state $\mathbf{h}^{t-1}$ and a new input $\mathbf{x}^t$.

Long short term memory (LSTM) [98] is considered one of the most successful RNN architectures. The original LSTM model can be neatly defined with the following equations:

$$\mathbf{i}^t = \sigma\left(\mathbf{W}_{ii}\mathbf{x}^t + \mathbf{W}_{ih}\mathbf{h}^{t-1} + \mathbf{b}_i\right) \tag{2}$$

$$\mathbf{f}^t = \sigma\left(\mathbf{W}_{fi}\mathbf{x}^t + \mathbf{W}_{fh}\mathbf{h}^{t-1} + \mathbf{b}_f\right) \tag{3}$$

$$\mathbf{c}^t = \mathbf{f}^t \odot \mathbf{c}^{t-1} + \mathbf{i}^t \odot \texttt{Tanh}\left(\mathbf{W}_{ci}\mathbf{x}^t + \mathbf{W}_{ch}\mathbf{h}^{t-1} + \mathbf{b}_c\right) \tag{4}$$

$$\mathbf{o}^t = \sigma\left(\mathbf{W}_{oi}\mathbf{x}^t + \mathbf{W}_{oh}\mathbf{h}^{t-1} + \mathbf{b}_o\right) \tag{5}$$

$$\mathbf{h}^t = \mathbf{o}^t \odot \texttt{Tanh}\left(\mathbf{c}^t\right) \tag{6}$$

Here $\mathbf{i}^t$, $\mathbf{f}^t$, and $\mathbf{o}^t$ represent the input, forget and output gates respectively, while $\mathbf{c}^t$ is the memory cell and $\mathbf{h}^t$ is the hidden state. σ and $\texttt{Tanh}$ represent the sigmoid and hyperbolic tangent activation functions respectively. Gated recurrent units (GRUs) [44] is another successful RNN architecture.

Fully attentive models have recently demonstrated on-par or superior performance compared to RNN variants for a variety of tasks (see, e.g., [222, 60, 126, 203]). These models rely only on (self-)attention and abstain from using recurrence. Vaswani *et al.* [222] characterize a self-attention mechanism as a function from query, key, and value vectors to a vector that is a weighted sum of the value vectors. Their mechanism is presented in Equation (7).

$$\begin{aligned}\texttt{Attention}(\mathbf{Q},\mathbf{K},\mathbf{V}) &= \texttt{softmax}(\frac{\mathbf{QK'}}{\sqrt{d_k}})\mathbf{V} \qquad (7)\\ \textit{where} \quad \mathbf{Q} = \mathbf{XW}_Q, \mathbf{K} &= \mathbf{XW}_K, \mathbf{V} = \mathbf{XW}_V\end{aligned}$$

where $\mathbf{Q} \in \mathbb{R}^{T\times d_k}$, $\mathbf{K} \in \mathbb{R}^{T\times d_k}$, $\mathbf{V} \in \mathbb{R}^{T\times d_v}$ are called the query, key and value matrices, $\mathbf{K'}$ is the transpose of $\mathbf{K}$, $\mathbf{X} \in \mathbb{T}\times$ is the input sequence, $\mathbf{W}_Q \in \mathbb{R}^{d\times d_k}$, $\mathbf{W}_K \in \mathbb{R}^{d\times d_k}$ and $\mathbf{W}_V \in \mathbb{R}^{d\times d_v}$ are weight matrices, and `softmax` performs a row-wise normalization of the input matrix. A mask is added to Equation (7) to make sure that at time T, the mechanism only allows a sequence model to attend to the points before time T. Vaswani *et al.* [222] also define a *multi-head* self-attention mechanism by considering multiple self-attention blocks (as defined in Equation (7)) each having different weight matrices and then concatenating the results.

2.7 Temporal Point Processes

Temporal Point Processes (TPP) [47] are stochastic, or random, processes that are used for modeling sequential asynchronous discrete events occurring in continuous time. Asynchronous in this context means that the time between consecutive events may not be the same. TPPs have been applied for applications like e-commerce [237], finance [8], etc. A typical realization of a TPP is a sequence of discrete events occurring at time points $t_1, t_2, t_3, \ldots$ for $t_i \leq T$, where the sequence has been generated by some stochastic process and T represents the time horizon of the process. A TPP model uses a conditional density function $\texttt{f}(t|\mathcal{H}_{t_n})$ indicating the density of the occurrence of the next event at some time point $t_n < t \leq T$ given the history $\mathcal{H}_{t_n}$ of the process till time t_n (including time t_n). The cumulative density function till time $t \geq t_n$ given the history $\mathcal{H}_{t_n}$ is defined as follows:

$$\mathcal{F}(t|\mathcal{H}_{t_n}) = \int_{\tau=t_n}^{t} \texttt{f}(\tau|\mathcal{H}_{t_n})d\tau \qquad (8)$$

Equation (8) also corresponds to the probability that the next event will happen between t_n and t. The survival function of a process [1] indicates the probability that no event will occur until $t \geq t_n$ given the history $\mathcal{H}_{t_n}$ and is computed as $\texttt{S}(t|\mathcal{H}_{t_n}) = 1 - \mathcal{F}(t|\mathcal{H}_{t_n})$. Having the density function, the time for the next event can be predicted by taking an expectation over $\texttt{f}(t|\mathcal{H}_{t_n})$ as:

$$\hat{t} \;=\; \mathop{\mathbb{E}}_{t\sim\texttt{f}(t|\mathcal{H}_{t_n})}[t] = \int_{\tau=t_n}^{T} \tau\texttt{f}(\tau|\mathcal{H}_{t_n})d\tau \qquad (9)$$

The parameters of a TPP can be learned from data by maximizing the joint density of the entire process defined as follows:

$$\texttt{f}(t_1,\ldots,t_n) = \prod_{i=1}^{n} \texttt{f}(t_i|\mathcal{H}_{t_{i-1}}) \qquad (10)$$

Another way of characterizing a TPP is through a conditional intensity function (a.k.a. hazard function) $\lambda(t \mid \mathcal{H}_{t-})$ such that $\lambda(t \mid \mathcal{H}_{t-})dt$ represents the probability of the occurrence of an event in the interval $[t, t+dt]$ given that no event has occurred until time $t_n < t \leq T$. $\mathcal{H}_{t-}$ represents the history of the process until t but not including t. The intensity and density functions can be derived from each other as follows:

$$\begin{aligned}\lambda(t|\mathcal{H}_{t-})dt &= \texttt{Prob}(t_{n+1} \in [t, t+dt] \mid \mathcal{H}_{t-})\\ &= \texttt{Prob}(t_{n+1} \in [t, t+dt] \mid \mathcal{H}_{t_n}, t_{n+1} \notin (t_n, t))\\ &= \frac{\texttt{Prob}(t_{n+1} \in [t, t+dt],\ t_{n+1} \notin (t_n, t) \ \mid\ \mathcal{H}_{t_n})}{\texttt{Prob}(t_{n+1} \notin (t_n, t) \ \mid\ \mathcal{H}_{t_n})}\\ &= \frac{\texttt{Prob}(t_{n+1} \in [t, t+dt] \ \mid\ \mathcal{H}_{t_n})}{\texttt{Prob}(t_{n+1} \notin (t_n, t) \ \mid\ \mathcal{H}_{t_n})}\\ &= \frac{\texttt{f}(t \mid \mathcal{H}_{t_n})dt}{\texttt{S}(t \mid \mathcal{H}_{t_n})} \qquad (11)\end{aligned}$$

The intensity function can be designed according to the application. The function usually contains learnable parameters [64] that can be learned from the data.

Example 5. Consider the problem of predicting when the next earthquake will occur in a region based on the times of previous earthquakes in that region. Typically, an earthquake is followed by a series of other earthquakes as aftershocks. Thus, upon observing an earthquake, a model should increase the probability of another earthquake in near future and gradually decay this probability.

Let $t_1, t_2, \ldots, t_n$ be the times at which an earthquake occurred in the region. Equation (12) gives one possible conditional intensity function for modeling this process.

$$\lambda^*(t) = \mu + \alpha \sum_{t_i \leq t} \exp(-(t - t_i)) \tag{12}$$

where μ and α are parameters that are constrained to be positive and are generally learned from the data. The sum is over all the timestamps $t_i < t$ at which an earthquake occurred. In this function, μ can be considered as the base intensity of an earthquake in the region. The occurrence of an earthquake increases the intensity of another earthquake in the near future (as it makes the value of the sum increase), which decays exponentially to the base intensity. The amount of increase is controlled by α. Note that the conditional intensity function is always positive as μ, α and $\exp(.)$ are always positive. From Equation 11, the density function for random variable t is $f(t|\mathcal{H}_{t_n}) = \lambda^*(t) * S(t|\mathcal{H}_{t_n})$. We can estimate the time for the occurrence of the next earthquake ($\hat{t}$) by taking an expectation over the random variable t as in Equation (9).

Equation (12) is a special case of the well-known self-exciting Hawkes process [92, 157]. Other well-studied TPPs include Poisson processes [122], self-correcting processes [108], and autoregressive conditional duration processes [71]. Depending on the application, one may use one of these intensity functions or even potentially design new ones. Recently, there has been growing interest in learning the intensity function entirely from the data [64].

3 Representation Learning for Static Graphs

In this section, we provide an overview of representation learning approaches for static graphs. The main aim of this section is to provide enough information for the descriptions and discussions in the next sections on dynamic graphs. Readers interested in learning more about representation learning on static graphs can refer to several existing surveys specifically written on this topic (e.g., see [88, 251, 27, 48] for graphs and [172, 227] for KGs).

3.1 Decoders

Assuming an encoder has provided the embedding function, the decoder aims at using the node and relation embeddings for node classification, edge prediction, graph classification, or other prediction purposes. We divide the discussion on decoders for static graphs into those used for graphs and those used for KGs.

3.1.1 Decoders for Static Graphs

For static graphs, the embedding function usually maps each node to a single vector; that is, $\texttt{EMB}(\mathsf{v}) = (\mathbf{z}_\mathsf{v})$ where $\mathbf{z}_\mathsf{v} \in \mathbb{R}^d$ for any $\mathsf{v} \in \mathcal{V}$. To classify a node v, a decoder can be any classifier on $\mathbf{z}_\mathsf{v}$ (e.g., logistic regression or random forest).

To predict a link between two nodes v and u, for undirected (and bipartite) graphs, the most common decoder is based on the *dot-product* of the vectors for the two nodes, i.e., $\mathbf{z}'_\mathsf{v}\mathbf{z}_\mathsf{u}$. The dot-product gives a score that can then be fed into a sigmoid function whose output can be considered as the probability of a link existing between v and u. Grover and Leskovec [83] propose several other decoders for link prediction in undirected graphs. Their decoders are based on defining a function $\texttt{f}(\mathbf{z}_\mathsf{v}, \mathbf{z}_\mathsf{u})$ that combines the two vectors $\mathbf{z}_\mathsf{v}$ and $\mathbf{z}_\mathsf{u}$ into a single vector. The resulting vector is then considered as the edge features that can be fed into a classifier to predict if an edge exists between v and u or not. These combining functions include:

- The average of the two vectors: $\frac{\mathbf{z}_\mathsf{v}+\mathbf{z}_\mathsf{u}}{2}$,

- The element-wise (Hadamard) multiplication of the two vectors: $\mathbf{z}_\mathsf{v} \odot \mathbf{z}_\mathsf{u}$,
- The element-wise absolute value of the difference of the two vectors: $\texttt{abs}(\mathbf{z}_\mathsf{v} - \mathbf{z}_\mathsf{u})$,
- The element-wise squared value of the difference of the two vectors: $(\mathbf{z}_\mathsf{v} - \mathbf{z}_\mathsf{u})^2$.

Instead of computing the distance between $\mathbf{z}_\mathsf{v}$ and $\mathbf{z}_\mathsf{u}$ in the Euclidean space, the distance can be computed in other spaces such as the hyperbolic space [32]. Different spaces offer different properties. Note that all these four combination functions are symmetric, i.e., $\texttt{f}(\mathbf{z}_\mathsf{v}, \mathbf{z}_\mathsf{u}) = \texttt{f}(\mathbf{z}_\mathsf{u}, \mathbf{z}_\mathsf{v})$ where $\texttt{f}$ is any of the above functions. This is an important property when the graph is undirected.

For link prediction in directed graphs, it is important to treat the source and target of the edge differently. Towards this goal, one approach is to *concatenate* the two vectors as $[\mathbf{z}_\mathsf{v}; \mathbf{z}_\mathsf{u}]$ and feed the concatenation into a classifier (see, e.g., [179]). Another approach used in [151] is to project the source and target vectors to another space as $\hat{\mathbf{z}}_\mathsf{v} = \mathbf{W}_1\mathbf{z}_\mathsf{v}$ and $\hat{\mathbf{z}}_\mathsf{u} = \mathbf{W}_2\mathbf{z}_\mathsf{u}$, where $\mathbf{W}_1$ and $\mathbf{W}_2$ are matrices with learnable parameters, and then take the dot-product in the new space (i.e., $\hat{\mathbf{z}}_\mathsf{v}'\hat{\mathbf{z}}_\mathsf{u}$). A third approach is to take the vector representation $\mathbf{z}_\mathsf{v}$ of a node $\mathsf{v} \in \mathcal{V}$ and send it through a feed-forward neural network with $|\mathcal{V}|$ outputs where each output gives the score for whether v has a link with one of the nodes in the graph or not. This approach is used mainly in graph autoencoders (see, e.g., [226, 28, 215, 81, 39]) and is used for both directed and undirected graphs.

The decoder for a graph classification task needs to compress node representations into a single representation which can then be fed into a classifier to perform graph classification. Duvenaud *et al.* [67] simply average all the node representations into a single vector. Gilmer *et al.* [80] consider the node representations of the graph as a set and use the DeepSet aggregation [250] to get a single representation. Li *et al.* [140] add a virtual node to the graph which is connected to all the nodes and use the representation of the virtual node as the representation of the graph. Several approaches perform a deterministic hierarchical graph clustering step and combine the node representations in each cluster to learn hierarchical representations [59, 75, 204]. Instead of performing a deterministic clustering and then running a graph classification model, Ying *et al.* [244] learn the hierarchical structure jointly with the classifier in an end-to-end fashion.

3.1.2 Decoders for Link Prediction in Static KGs

There are several classes of decoders for link prediction in static KGs. Here, we provide an overview of the *translational*, *bilinear*, and *deep learning* classes. When we discuss the expressivity of the decoders in this subsection, we assume the decoder is combined with a flexible encoder.

Translational decoders usually assume the encoder provides an embedding function such that $\texttt{EMB}(\mathsf{v}) = (\mathbf{z}_\mathsf{v})$ for every $\mathsf{v} \in \mathcal{V}$ where $\mathbf{z}_\mathsf{v} \in \mathbb{R}^{d_1}$, and $\texttt{EMB}(\mathsf{r}) = (\mathbf{z}_\mathsf{r}, \mathbf{P}_\mathsf{r}, \mathbf{Q}_\mathsf{r})$ for every $\mathsf{r} \in \mathcal{R}$ where $\mathbf{z}_\mathsf{r} \in \mathbb{R}^{d_2}$, $\mathbf{P}_\mathsf{r} \in \mathbb{R}^{d_1 \times d_2}$, and $\mathbf{Q}_\mathsf{r} \in \mathbb{R}^{d_1 \times d_2}$. That is, the embedding for a node contains a single vector whereas the embedding for a relation contains a vector and two matrices. For an edge $(\mathsf{v}, \mathsf{r}, \mathsf{u})$, these models use:

$$||\mathbf{P}_\mathsf{r}\mathbf{z}_\mathsf{v} + \mathbf{z}_\mathsf{r} - \mathbf{Q}_\mathsf{r}\mathbf{z}_\mathsf{u}||_i \tag{13}$$

as the dissimilarity score for the edge where $||.||_i$ represents norm i of a vector. i is usually either 1 or 2. Translational decoders differ in the restrictions they impose on $\mathbf{P}_\mathsf{r}$ and $\mathbf{Q}_\mathsf{r}$. TransE [17] constrains $\mathbf{P}_\mathsf{r} = \mathbf{Q}_\mathsf{r} = \mathbf{I_d}$. So the dissimilarity function for TransE can be simplified to:

$$||\mathbf{z}_\mathsf{v} + \mathbf{z}_\mathsf{r} - \mathbf{z}_\mathsf{u}||_i \tag{14}$$

In TransR [147], $P_r = Q_r$. In STransE [168], no restrictions are imposed on the matrices. Kazemi and Poole [115] proved that TransE, TransR, STransE, and many other variants of translational approaches are not fully expressive with respect to link prediction (regardless of the encoder) and identified severe restrictions on the type of relations that can be modeled using these approaches.

Bilinear decoders usually assume the encoder provides an embedding function such that $\texttt{EMB}(\mathsf{v}) = (\mathbf{z}_\mathsf{v})$ for every $\mathsf{v} \in \mathcal{V}$ where $\mathbf{z}_\mathsf{v} \in \mathbb{R}^d$, and $\texttt{EMB}(\mathsf{r}) = (\mathbf{P}_\mathsf{r})$ for every $\mathsf{r} \in \mathcal{R}$ where $\mathbf{P}_\mathsf{r} \in \mathbb{R}^{d \times d}$. For an edge $(\mathsf{v}, \mathsf{r}, \mathsf{u})$, these models use:

$$\mathbf{z}_\mathsf{v}'\mathbf{P}_\mathsf{r}\mathbf{z}_\mathsf{u} \tag{15}$$

as the similarity score for the edge. Bilinear decoders differ in the restrictions they impose on $\mathbf{P}_\mathsf{r}$ matrices [229]. In RESCAL [171], no restrictions are imposed on the $\mathbf{P}_\mathsf{r}$ matrices. RESCAL is fully

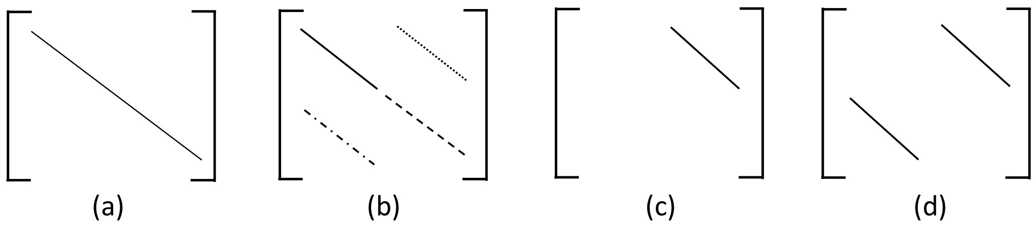

Figure 2: A graphical representation of the constraints over the P_r matrices for bilinear models (a) DistMult, (b) ComplEx, (c) CP, and (d) SimplE taken from [115, 118] where lines represent the non-zero elements of the matrices (other elements are constrained to be zero). In ComplEx, the parameters represented by the dashed line are tied (i.e., equal) to the parameters represented by the solid line and the parameters represented by the dotted line are tied to the negative of the dotted-and-dashed line.

expressive with respect to link prediction, but the large number of parameters per relation makes RESCAL prone to overfitting. To reduce the number of parameters in RESCAL, DistMult [241] constrains the $\mathbf{P}_\mathsf{r}$ matrices to be diagonal. This reduction in the number of parameters, however, comes at a cost: DistMult loses expressivity and is only able to model symmetric relations. That is because the score function of DistMult does not distinguish between the source and target vectors.

ComplEx [218], CP [97] and SimplE [115] reduce the number of parameters in RESCAL without sacrificing expressivity. ComplEx extends DistMult by assuming the embeddings are complex (instead of real) valued, i.e. $\mathbf{z}_\mathsf{v} \in \mathbb{C}^d$ and $\mathbf{P}_\mathsf{r} \in \mathbb{C}^{d\times d}$ for every $\mathsf{v} \in \mathcal{V}$ and $\mathsf{r} \in \mathcal{R}$. Then, it slightly changes the score function to $\texttt{Real}(\mathbf{z}'_\mathsf{v}\mathbf{P}_\mathsf{r}\texttt{conjugate}(\mathbf{z}_\mathsf{u}))$ where `Real` returns the real part of an imaginary number and `conjugate` takes an element-wise conjugate of the vector elements. By taking the conjugate of the target vector, ComplEx differentiates between source and target nodes and does not suffer from the symmetry issue of DistMult. CP defines $\texttt{EMB}(\mathsf{v}) = (\mathbf{z}_\mathsf{v}, \overline{\mathbf{z}}_\mathsf{v})$, i.e. the embedding of a node consists of two vectors, where $\mathbf{z}_\mathsf{v}$ captures the v's behaviour when it is the source of an edge and $\overline{\mathbf{z}}_\mathsf{v}$ captures v's behaviour when it is the target of an edge. For relations, CP defines $\texttt{EMB}(\mathsf{r}) = (\mathbf{z}_\mathsf{r})$. The similarity function of CP for an edge $(\mathsf{v}, \mathsf{r}, \mathsf{u})$ is then defined as $\mathbf{z}'_\mathsf{v}\texttt{diag}(\mathbf{z}_\mathsf{r})\overline{\mathbf{z}}_\mathsf{u}$. Realizing the information may not flow well between the two vectors of a node, SimplE adds another vector to the relation embeddings as $\texttt{EMB}(\mathsf{r}) = (\mathbf{z}_\mathsf{r}, \overline{\mathbf{z}}_\mathsf{r})$ where $\overline{\mathbf{z}}_\mathsf{r}$ models the behaviour of the inverse of the relation. Then, it changes the score function to be the average of $\mathbf{z}'_\mathsf{v}\texttt{diag}(\mathbf{z}_\mathsf{r})\overline{\mathbf{z}}_\mathsf{u}$ and $\overline{\mathbf{z}}'_\mathsf{u}\texttt{diag}(\overline{\mathbf{z}}_\mathsf{r})\mathbf{z}_\mathsf{v}$.

For ComplEx, CP, and SimplE, it is possible to view the embedding for each node v as a single vector in $\mathbb{R}^{2d}$ by concatenating the two vectors (in the case of ComplEx, the two vectors correspond to the real and imaginary part of the embedding vector). Then, the $\mathbf{P}_\mathsf{r}$ matrices can be viewed as being restricted according to Figure 2 (taken from [115]).

Other bilinear approaches include HolE [194] whose equivalence to ComplEx has been established [93], and Analogy [148] where the $\mathbf{P}_\mathsf{r}$ matrices are constrained to be block-diagonal.

Deep learning-based decoders: Deep learning approaches typically use feed-forward or convolutional neural networks for scoring edges in a KG. Dong *et al.* [63] and Santoro *et al.* [197] consider $\texttt{EMB}(\mathsf{v}) = (\mathbf{z}_\mathsf{v})$ for every node $\mathsf{v} \in \mathcal{V}$ such that $\mathbf{z}_\mathsf{v} \in \mathbb{R}^{d_1}$ and $\texttt{EMB}(\mathsf{r}) = (\mathbf{z}_\mathsf{r})$ for every relation $\mathsf{r} \in \mathcal{R}$ such that $\mathbf{z}_\mathsf{r} \in \mathbb{R}^{d_2}$. Then for an edge $(\mathsf{v}, \mathsf{r}, \mathsf{u})$, they feed $[\mathbf{z}_\mathsf{v}; \mathbf{z}_\mathsf{r}; \mathbf{z}_\mathsf{u}]$ (i.e., the concatenation of the three vector representations) into a feed-forward neural network that outputs a score for this edge. Dettmers *et al.* [61] develop a score function based on convolutions. They consider $\texttt{EMB}(\mathsf{v}) = (\mathbf{Z}_\mathsf{v})$ for each node $\mathsf{v} \in \mathcal{V}$ such that $\mathbf{Z}_\mathsf{v} \in \mathbb{R}^{d_1\times d_2}$ and $\texttt{EMB}(\mathsf{r}) = (\mathbf{Z}_\mathsf{r})$ for each relation $\mathsf{r} \in \mathcal{R}$ such that $\mathbf{Z}_\mathsf{r} \in \mathbb{R}^{d_1\times d_2}$[1]. For an edge $(\mathsf{v}, \mathsf{r}, \mathsf{u})$, first they combine $\mathbf{Z}_\mathsf{v}$ and $\mathbf{Z}_\mathsf{r}$ into a matrix $\mathbf{Z}_{vr} \in \mathbb{R}^{2d_1\times d_2}$ by concatenating the two matrices on the rows, or by adding the i^{th} row of each matrix in turn. Then 2D convolutions with learnable filters are applied on $\mathbf{Z}_\mathsf{vr}$ generating multiple matrices and the matrices are vectorized into a vector $\mathbf{c}_\mathsf{vr} \in \mathbb{R}^l$, where the size l of the vector depends on the number of convolution filters. Then the score for the edge is computed as:

$$(\mathbf{c}'_\mathsf{vr}\mathbf{W})\texttt{vec}(\mathbf{Z}_\mathsf{u}) \tag{16}$$

[1]Alternatively, the matrices can be viewed as vectors of size $d_1 d_2$.

where $\mathbf{W} \in \mathbb{R}^{l \times (d_1 d_2)}$ is a weight matrix. Other deep learning approaches include [11] which is another score function based on convolutions, and [206] which contains feed-forward components as well as several bilinear components.

3.2 Encoders

In the previous section, we discussed how an embedding function can be used by a decoder to make predictions. In this section, we describe different approaches for creating encoders that provide the embedding function to be consumed by the decoder.

3.2.1 High-Order Proximity Matrices

While the adjacency matrix of a graph only represents local proximities, one can also define *high-order proximity* matrices [176] or *similarity metrics* [49]. Let $\mathbf{S}$ be a high-order proximity matrix. A simple approach for creating an encoder is to let $\mathtt{EMB}(\mathsf{v}_i) = (\mathbf{S}[i])$ (or $\mathtt{EMB}(\mathsf{v}_i) = (\mathbf{S}'[i])$) corresponding to the i^{th} row (or the i^{th} column) of matrix $\mathbf{S}$. Encoders based on high-order proximity matrices are typically parameter-free and do not require learning (although some of them have hyper-parameters that need to be tuned). In what follows, we describe several of these matrices.

- *Common neighbours* matrix is defined as $\mathbf{S}_{CN} = \mathbf{AA}$. $\mathbf{S}_{CN}[i][j]$ corresponds to the number of nodes that are connected to both v_i and v_j. For a directed graph, $\mathbf{S}_{CN}[i][j]$ counts how many nodes v are simultaneously the target of an edge starting at v_i and the source of an edge ending at v_j.
- *Jaccard's coefficient* is a slight modification of $\mathbf{S}_{CN}$ where one divides the number of common neighbours of v_i and v_j by the total number of distinct nodes that are the targets of edges starting at v_i or the sources of edges ending at v_j. Formally, Jaccard's coefficient is defined as $\mathbf{S}_{JC}[i][j] = \mathbf{S}_{CN}[i][j] / (\sum_{k=1}^{N} (\mathbf{A}[i][k] + \mathbf{A}[k][j] - \mathbf{S}_{CN}[i][j]))$.
- *Adamic-Adar* is defined as $\mathbf{S}_{AA} = \mathbf{A}\hat{\mathbf{D}}\mathbf{A}$, where $\hat{\mathbf{D}}[i][i] = 1/\sum_{j=1}^{N}(\mathbf{A}[i][j] + \mathbf{A}[j][i])$. $\mathbf{S}_{AA}$ computes the weighted sum of common neighbours where the weight is inversely proportional to the degree of the neighbour.
- *Katz index* is defined as $\mathbf{S}_{Katz} = \sum_{j=1}^{\infty} (\beta \mathbf{A})^j$ computes a weighted sum of all the paths between two nodes v_i and v_j. β controls the depth of the connections: the closer β is to 1, the longer paths one wants to consider. One can rewrite the formula recursively as $\beta \mathbf{A} \mathbf{S}_{Katz} + \beta \mathbf{A} = \mathbf{S}_{Katz}$ and, as a corollary, obtain $\mathbf{S}_{Katz} = (\mathbf{I}_N - \beta \mathbf{A})^{-1} \beta \mathbf{A}$.
- *Preferential Attachment* is simply a product of in- and out- degrees of nodes: $\mathbf{S}_{PA}[i][j] = (\sum_{k=1}^{N} \mathbf{A}[i][k])(\sum_{k=1}^{N} \mathbf{A}[k][j])$.

3.2.2 Shallow Encoders

Shallow encoders first decide on the number and the shape of the vectors and matrices for node and relation embeddings. Then, they consider each element in these vectors and matrices as a parameter to be directly learned from the data. As an example, consider the problem of link prediction in a KG. Let the encoder be a shallow encoder with $\mathtt{EMB}(\mathsf{v}) = (\mathbf{z}_\mathsf{v})$ for each node v in the KG and $\mathtt{EMB}(\mathsf{r}) = (\mathbf{P}_\mathsf{r})$ for each relation r in the KG, and the decoder be the RESCAL function. $\mathbf{z}_\mathsf{v}$'s and $\mathbf{P}_\mathsf{r}$'s are initialized randomly and then their values are optimized such that $\mathbf{z}_\mathsf{v}' \mathbf{P}_\mathsf{r} \mathbf{z}_\mathsf{u}$ becomes a large positive number if $(\mathsf{v}, \mathsf{r}, \mathsf{u})$ is in positive samples and $\mathbf{z}_\mathsf{v}' \mathbf{P}_\mathsf{r} \mathbf{z}_\mathsf{u}$ becomes a large negative number if $(\mathsf{v}, \mathsf{r}, \mathsf{u})$ is in negative samples.

3.2.3 Decomposition Approaches

Decomposition methods are among the earliest attempts for developing encoders for graphs. They learn node embeddings similar to shallow encoders but in an unsupervised way: the node embeddings are learned in a way that connected nodes are close to each other in the embedded space. Once the embeddings are learned, they can be used for purposes other than reconstructing the edges (e.g., for clustering). Formally, for an undirected graph $\mathcal{G}$, learning node embeddings $\mathtt{EMB}(\mathsf{v}_i) = (\mathbf{z}_{\mathsf{v}_i})$, where $\mathbf{z}_{\mathsf{v}_i} \in \mathbb{R}^d$, such that connected nodes are close in the embedded space can be done through solving the following optimization problem:

$$\min_{\{\mathbf{z}_{\mathsf{v}_i}\}_{i=1}^{N}} \sum_{i,j} \mathbf{A}[i][j]||\mathbf{z}_{\mathsf{v}_i} - \mathbf{z}_{\mathsf{v}_j}||^2 \tag{17}$$

This loss ensures that connected nodes are close to each other in the embedded space. One needs to impose some constraints to get rid of a scaling factor and to eliminate the trivial solution where all nodes are set to a single vector. For that let us consider a new matrix $\mathbf{Y} \in \mathbb{R}^{n \times d}$, such that its rows give the embedding: $\mathbf{Y}[i] = \mathbf{z}'_{\mathsf{v}_i}$. Then one can add the constraints to the optimization problem (17): $\mathbf{Y}'\mathbf{D}\mathbf{Y} = \mathbf{I}$, where $\mathbf{D}$ is a diagonal matrix of degrees as defined in Subsection 2.1. As was proved in [13], this constrained optimization is equivalent to solving a generalized eigenvalue decomposition:

$$\mathbf{L}\mathbf{y} = \lambda\mathbf{D}\mathbf{y}, \tag{18}$$

where $\mathbf{L}$ is a graph Laplacian; and the matrix $\mathbf{Y}$ can be obtained by considering the $N \times d$ matrix of top-d generalized eigenvectors: $\mathbf{Y} = [\mathbf{y}_1 \dots \mathbf{y}_d]$.

Sussman *et al.* [211] suggested to use a slightly different embedding based on the eigenvalue decomposition of the adjacency matrix $\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{U}'$ (this matrix is symmetric for an undirected graph). Then one can choose the top d eigenvalues $\{\lambda_1, \dots, \lambda_d\}$ and the corresponding eigenvectors $\{\mathbf{u}_1, \dots, \mathbf{u}_d\}$ and construct a new matrix

$$\mathbf{Z} = \mathbf{U}_{<d}\sqrt{\mathbf{\Sigma}_{<d}} \in \mathbb{R}^{N \times d}, \tag{19}$$

where $\mathbf{\Sigma}_{<d} = \text{diag}(\lambda_1, \dots, \lambda_d)$, and $\mathbf{U}_{<d} = [\mathbf{u}_1 \dots \mathbf{u}_d]$. Rows of this matrix can be used as node embedding: $\mathbf{z}_{\mathsf{v}_i} = \mathbf{Z}[i]' \in \mathbb{R}^d$. This is the so called *adjacency spectral embedding*, see also [139].

For directed graphs, because of their asymmetric nature, keeping track of the n^{th}-order neighbours where $n > 1$ becomes difficult. For this reason, working with a high-order proximity matrix $\mathbf{S}$ is preferable. Furthermore, for directed graphs, it may be preferable to learn two vector representations per node, one to be used when the node is the source and the other to be used when the node is the target of an edge. One may learn embeddings for directed graphs by solving the following:

$$\min_{\mathbf{Z}_s, \mathbf{Z}_t} \; ||\mathbf{S} - \mathbf{Z}_s\mathbf{Z}_t'||_F^2, \tag{20}$$

where $||.||_F$ is the Frobenius norm and $\mathbf{Z}_s, \mathbf{Z}_t \in \mathbb{R}^{N \times d}$. Given the solution, one can define the "source" features of a node v_i as $\mathbf{Z}_s[i]'$ and the "target" features as $\mathbf{Z}_t[i]'$. A single-vector embedding of a node v_i can be defined as a concatenation of these features. The Eckart–Young–Mirsky theorem [69] from linear algebra indicates that the solution is equivalent to finding the singular value decomposition of $\mathbf{S}$:

$$\mathbf{S} = \mathbf{U}_s\mathbf{\Sigma}(\mathbf{U}_t)', \tag{21}$$

where $\mathbf{\Sigma} = \text{diag}(\sigma_1, \dots, \sigma_N)$ is a matrix of singular values and $\mathbf{U}_s$ and $\mathbf{U}_t$ are matrices of left and right singular vectors respectively (stacked as columns). Then using the top d singular vectors one gets the solution of the optimization problem in (20):

$$\mathbf{Z}_s = (\mathbf{U_s})_{<d}\sqrt{\mathbf{\Sigma}}_{<d} \tag{22}$$

$$\mathbf{Z}_t = (\mathbf{U_t})_{<d}\sqrt{\mathbf{\Sigma}}_{<d}. \tag{23}$$

3.2.4 Random Walk Approaches

One of the popular classes of approaches for learning an embedding function for graphs is the class of random walk approaches. Similar to decomposition approaches, encoders based on random walks also learn embeddings in an unsupervised way. However, compared to decomposition approaches, these embeddings may capture longer term dependencies. To describe the encoders in this category, first we define what a random walk is and then describe the encoders that leverage random walks to learn an embedding function.

Definition 7. A *random walk* for a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is a sequence of nodes $\mathsf{v}_1, \mathsf{v}_2, \dots, \mathsf{v}_l$ where $\mathsf{v}_i \in \mathcal{V}$ for all $1 \leq i \leq l$ and $(\mathsf{v}_i, \mathsf{v}_{i+1}) \in \mathcal{E}$ for all $1 \leq i \leq l - 1$. l is called the *length* of the walk.

A random walk of length l can be generated by starting at a node v_i in the graph, then transitioning to a neighbor v_j of v_i ($j \neq i$), then transitioning to a neighbor of v_j and continuing this process for l steps. The selection of the first node and the node to transition to in each step can be uniformly at random or based on some distribution/strategy.

Example 6. Consider the graph in Figure 1(b). The following are three examples of random walks on this graph with length 4:

$$1)\ \mathsf{v}_1, \mathsf{v}_3, \mathsf{v}_2, \mathsf{v}_3 \qquad 2)\ \mathsf{v}_2, \mathsf{v}_1, \mathsf{v}_2, \mathsf{v}_4 \qquad 3)\ \mathsf{v}_4, \mathsf{v}_2, \mathsf{v}_4, \mathsf{v}_2$$

In the first walk, the initial node has been selected to be v_1. Then a transition has been made to v_3, which is a neighbor of v_1. Then a transition has been made to v_2, which is a neighbor of v_3 and then a transition back to v_3, which is a neighbor of v_2. The following are two examples of invalid random walks:

$$1)\ \mathsf{v}_1, \mathsf{v}_4, \mathsf{v}_2, \mathsf{v}_3 \qquad 2)\ \mathsf{v}_1, \mathsf{v}_3, \mathsf{v}_4, \mathsf{v}_2$$

The first one is not a valid random walk since a transition has been made from v_1 to v_4 when there is no edge between v_1 and v_4. The second one is not valid because a transition has been made from v_3 to v_4 when there is no edge between v_3 and v_4.

Random walk encoders perform multiple random walks of length l on a graph and consider each walk as a sentence, where the nodes are considered as the words of these sentences. Then they use the techniques from natural language processing for learning word embeddings (e.g., [161, 182]) to learn a vector representation for each node in the graph. One such approach is to create a matrix $\mathbf{S}$ from these random walks such that $\mathbf{S}[i][j]$ corresponds to the number of times v_i and v_j co-occurred in random walks and then factorize the matrix (see Section 3.2.3) to get vector representations for nodes.

Random walk encoders typically differ in the way they perform the walk, the distribution they use for selecting the initial node, and the transition distribution they use. For instance, DeepWalk [183] selects both the initial node and the node to transition to uniformly at random. Perozzi *et al.* [184] extends DeepWalk by allowing random walks to skip over multiple nodes at each transition. Node2Vec [83] selects the node to transition to based on a combination of breadth-first search (to capture local information) and depth-first search (to capture global information).

3.2.5 Autoencoder Approaches

Another class of models for learning an embedding function for static graphs is by using autoencoders. Similar to the decomposition approaches, these approaches are also unsupervised. However, instead of learning shallow embeddings that reconstruct the edges of a graph, the models in this category create a deep encoder that compresses a node's neighbourhood to a vector representation, which can be then used to reconstruct the node's neighbourhood. The model used for compression and reconstruction is referred to as an autoencoder. Similar to the decomposition approaches, once the node embeddings are learned, they may be used for purposes other than predicting a node's neighbourhood.

In its simplest form, an autoencoder [95] contains two components called the *encoder* and *decoder*, where each component is a feed-forward neural network. To avoid confusion with graph encoder and decoders, we refer to these two components as the first and second component. The first component takes as input a vector $\mathbf{a} \in \mathbb{R}^N$ (e.g., corresponding to N numerical features of an object) and passes it through several feed-forward layers producing another vector $\mathbf{z} \in \mathbb{R}^d$ such that $d << N$. The second component receives $\mathbf{z}$ as input and passes it through several feed-forward layers aiming at reconstructing $\mathbf{a}$. That is, assuming the output of the second component is $\hat{\mathbf{a}}$, the two components are trained such that $||\mathbf{a} - \hat{\mathbf{a}}||$ is minimized. $\mathbf{z}$ can be considered a compression of $\mathbf{a}$.

Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be a graph with adjacency matrix $\mathbf{A}$. For a node $\mathsf{v}_i \in \mathcal{V}$, let $\mathbf{A}[i]$ represent the i^{th} row of the adjacency matrix corresponding to the neighbors of v_i. To use autoencoders for generating node embeddings, Wang *et al.* [226] train an autoencoder (named *SDNE*) that takes a vector $\mathbf{A}[i]$ as input, compresses it to $\mathbf{z}_i$ in its first component, and then reconstructs it in its second component. After training, the $\mathbf{z}_i$ vectors corresponding to the output of the first component of the autoencoder can be considered as embeddings for the nodes v_i. $\mathbf{z}_i$ and $\mathbf{z}_j$ may further be constrained to be close in Euclidean space if v_i and v_j are connected. For the case of attributed graphs, Tran [215] concatenates the attribute values $\mathbf{x}_i$ of node v_i to $\mathbf{A}[i]$ and feeds the concatenation $[\mathbf{x}_i; \mathbf{A}[i]]$ into an autoencoder. Cao *et al.* [28] propose an autoencoder approach (named *RDNG*) that is similar to SDNE, but they first compute a similarity matrix $\mathbf{S} \in \mathbb{R}^{N \times N}$ based on two nodes co-occurring on random walks (any other matrix from Section 3.2.1 may also be used) showing the pairwise similarity of each pair of nodes, and then feed $\mathbf{S}[i]$s into the autoencoder.

3.2.6 Graph Convolutional Network Approaches

Yet another class of models for learning node embeddings in a graph are graph convolutional networks (GCNs). As the name suggests, graph convolutions generalize convolutions to arbitrary graphs. Graph convolutions have spatial (see, e.g., [87, 88, 199, 80]) and spectral constructions (see, e.g., [145, 123, 59, 138]). Here, we describe the spatial (or message passing) view and refer the reader to [23] for the spectral view.

A GCN consists of multiple layers where each layer takes node representations (a vector per node) as input and outputs transformed representations. Let $\mathbf{z}_{\mathsf{v},l}$ be the representation for a node v after passing it through the l^{th} layer. A very generic forward pass through a GCN layer transforms the representation of each node v as follows:

$$\mathbf{z}_{\mathsf{v},l+1} = \mathtt{transform}(\{\mathbf{z}_{\mathsf{v},j}\}_{0\le j\le l}, \{\mathbf{z}_{\mathsf{u},k}\}_{\mathsf{u}\in\mathcal{N}(\mathsf{v}),0\le k\le l}, \Theta) \tag{24}$$

where $\mathcal{N}(\mathsf{v})$ represents the neighbours of v and `transform` is a function parametrized by Θ which aggregates the information from the previous representations of the neighbours of v and combines it with the previous representations of v itself to compute $\mathbf{z}_{\mathsf{v},l+1}$. One of the key requirements of this function is that it should be invariant to the order of the nodes in $\mathcal{N}(\mathsf{v})$ because there is no specific ordering to nodes in an arbitrary graph. Another requirement for this function is that it should be able to handle a variable number of neighbours. If the graph is attributed, for each node v, $\mathbf{z}_{\mathsf{v},0}$ can be initialized to $\mathbf{x}_{\mathsf{v}}$ corresponding to the attribute values of v (see, e.g., [123]). Otherwise, they can be initialized using a one-hot encoding of the nodes (see, e.g.,[199]). In a GCN with L layers, each node receives information from the nodes at most L hops away from it (see Example 7).

Example 7. Consider the graph in Figure 1(b). To get $\mathbf{z}_{\mathsf{v}_1,1}$, corresponding to the node representation for v_1 after the first layer, according to Equation (24), the `transform` function aggregates the features of v_2 and v_3 and combines them with the features of v_1. $\mathbf{z}_{\mathsf{v}_2,1}, \mathbf{z}_{\mathsf{v}_3,1}$ and $\mathbf{z}_{\mathsf{v}_4,1}$ are computed similarly. Thus, after one step of applying Equation (24) (corresponding to one layer of a GCN), the representation of each node contains information about its 1^{st}-order neighbours (e.g., $\mathbf{z}_{\mathsf{v}_1,1}$ contains information about v_2 and v_3). Applying Equation (24) on v_1 for a second time encodes information from $\mathbf{z}_{\mathsf{v}_2,1}$ and $\mathbf{z}_{\mathsf{v}_3,1}$ into $\mathbf{z}_{\mathsf{v}_1,2}$. Since $\mathbf{z}_{\mathsf{v}_2,1}$ contains information about v_4, $\mathbf{z}_{\mathsf{v}_1,2}$ will contain information from v_1's 2^{nd}-order neighbour v_4. Extending this argument, the l^{th} layer of a GCN can be seen as integrating information from l^{th}-order neighbours.

There is a large literature on the design of the `transform` function (see, e.g., [140, 123, 87, 51]). Kipf and Welling [123] formulate it as:

$$\mathbf{Z}_{l+1} = \sigma(\tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}}\mathbf{Z}_l\mathbf{W}_{l+1}) \tag{25}$$

where $\tilde{\mathbf{A}} = \mathbf{A} + \mathbf{I}_N$ is adjacency matrix with self-connections for input graph, N is the number of nodes in the graph, $\mathbf{I}_N$ is the identity matrix, $\mathbf{W}_{l+1}$ is a parameter matrix for the $(l+1)^{\text{th}}$ layer and $\sigma(.)$ is a non-linearity. $\tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}}\mathbf{Z}_l$ corresponds to taking a normalized average of the features of v and its neighbours (treating the features of v and its neighbours identically). *GraphSage* [87] formulates the `transform` function as follows:

$$\mathbf{z}_{\mathsf{v},l+1} = \sigma(\mathbf{W}_{l+1}[\mathbf{z}_{\mathcal{N}(\mathsf{v})}; \mathbf{z}_{\mathsf{v},l}]) \tag{26}$$

$$where \quad \mathbf{z}_{\mathcal{N}(\mathsf{v})} = \mathtt{F}(\{\mathbf{z}_{\mathsf{u},l} | \mathsf{u} \in \mathcal{N}(\mathsf{v})\}) \tag{27}$$

where $\mathtt{F}(.)$ is either an element-wise mean/max operation or an LSTM taking as input the features of the neighbours with a random order. Compared to [123], GraphSage treats the features of v and its neighbours differently. Other formulations for the `transform` function can be found in several recent surveys (see, e.g., [256, 27]).

For a node v, not all the neighbouring nodes may be equally important. Some works aim at learning the importance of the neighbouring nodes in the `transform` function. Following the success of attention in sequence models [94, 222], Veličković *et al.* [223] propose an adaptive attention mechanism that learns to weigh the neighbours depending on their importance when aggregating information from the neighbours. The mechanism is adaptive in the sense that the weight of a node is not fixed and depends on the current representation of the node for which the aggregation is performed. Following Vaswani *et al.* [222], Veličković *et al.* [223] also use multi-headed attention. *GaAN* [252] extends this idea and introduces adaptive attention weights for different attention heads, i.e., the weights for different attention heads depend on the node for which the mulit-head attention is being applied.

In graphs like social networks, there can be nodes that have a large number of neighbours. This can make the `transform` function computationally prohibitive. Hamilton *et al.* [87] propose to use a uniform sampling of the neighbours so as to fix the neighbourhood size to a constant number. Not only the sampling helps reduce computational complexity and speed up training, but also it acts as a regularizer. Ying *et al.* [243] propose an extension of this idea according to which the neighbourhood of a node v is formed by repeatedly starting truncated random walks from v and choosing the nodes most frequently hit by these truncated random walks. In this way, the neighborhood of a node consists of the nodes most relevant to it, regardless of whether they are connected with an edge or not.

Xu *et al.* [238] study the expressiveness of certain GCN models with respect to graph classification and show that in terms of distinguishing different graphs, these GCNs are *at most* as powerful as the Weisfeiler-Lehman isomorphism test [231] — a test which is able to distinguish a broad class of graphs [7] but also known to fail in some corner cases [26]. They provide the necessary conditions under which these GCNs become as powerful as the Weisfeiler-Lehman test.

3.2.7 Encoders for KGs

For KG embedding, most existing approaches rely on shallow encoders (see e.g., [171, 241, 218, 17, 168, 115, 61]) with a few exceptions. One exception is relational GCNs (R-GCNs) [199]. The core operation that R-GCNs do differently is the application of a relation specific transformation (i.e., the transformation depends on the direction and the label of the edge) to the neighbors of the nodes in the aggregation function. In R-GCNs, the `transform` function is defined as follows:

$$\mathbf{z}_{\mathsf{v},l+1} = \sigma\left(\sum_{\mathsf{r}\in\mathcal{R}}\sum_{\mathsf{u}\in\mathcal{N}(\mathsf{v},\mathsf{r})}\frac{1}{c_{\mathsf{v},\mathsf{r}}}\mathbf{W}_{r,l}\mathbf{z}_{\mathsf{u},l} + \mathbf{W}_{0,l}\mathbf{z}_{\mathsf{v},l}\right) \tag{28}$$

where $\mathcal{R}$ is the set of all relation types in the KG, $\mathcal{N}(\mathsf{v},\mathsf{r})$ is the set of neighbouring nodes related to v via relation r, $c_{\mathsf{v},\mathsf{r}}$ is a normalization factor that can either be learned or fixed (e.g., to $|\mathcal{N}(\mathsf{v},\mathsf{r})|$), $\mathbf{W}_{\mathsf{r},l}$ is the transformation matrix for relation r at the l^{th} layer, and $\mathbf{W}_{0,l}$ is a self-transformation matrix at the l^{th} layer. Schlichtkrull *et al.* [199] initialize $\mathbf{z}_{\mathsf{v},0}$s as one hot vectors. If the graph is attributed, the $\mathbf{z}_{\mathsf{v},0}$s may be initialized using those attributes. Models using R-GCNs have a high capacity that allows them to encode information from l^{th} order neighbours explicitly by stacking l layers. However, stacking multiple layers increases the number of parameters quickly. Schlichtkrull *et al.* [199] propose two ways for keeping the number of parameters manageable. Sourek *et al.* [207] and Kazemi and Poole [114] propose other variants for Equation (28) where (roughly) the transformations are done using soft first-order logic rules.

4 Decoders for Dynamic Graphs

We divide the decoders for dynamic graphs into two categories: time-predicting decoders and time-conditioned decoders. In what follows, we explain each category and provide a summary of the existing approaches for that category.

4.1 Time-Predicting Decoders

The aim of the time-predicting decoders is two folds: 1- predicting what will happen in the future, 2- predicting when it will happen. For instance, they aim at predicting when *Bob* will visit *Montreal* (which is more informative than just predicting if *Bob* will visit *Montreal*).

Sun *et al.* [210] were among the first to study the problem of predicting when a particular type of relation will be formed between two nodes. To make such a prediction, first they find all paths between two nodes. These paths are matched with a set of pre-defined path templates and the number of paths matching each template is counted. These counts, which can be roughly considered as node-pair embeddings, are fed into a generalized linear model (GLM) and the score of this model is used to define the parameters of the density function. Sun *et al.* [210] use exponential, Weibull [230], and geometric distributions for defining the density function. Sun *et al.* [210] define the density function for the formation of a relation between two nodes for exponential distribution as follows:

$$\mathtt{f}(t) = \frac{1}{\theta}\exp\left(-\frac{t}{\theta}\right) \tag{29}$$

where θ is the output of the GLM model. An expectation of $t \sim f$ can be used to predict when the relation will be formed between the two nodes, as described in Section 2.7.

Recently there has been growing interest towards time predicting decoders [217, 216, 261]. Trivedi *et al.* [216] consider an encoder that provides an embedding function such that given a dynamic graph until time t gives $\mathtt{EMB}(\mathsf{v}) = (\mathbf{z}^t_\mathsf{v})$ and $\mathtt{EMB}(\mathsf{r}) = (\mathbf{P}_\mathsf{r})$. Trivedi *et al.* [216] first compute a score for the formation of a relation r between two nodes v and u as follows:

$$\mathtt{s}_{\mathsf{v},\mathsf{r},\mathsf{u}}(t) = \mathbf{z}^{t'}_\mathsf{v} \mathbf{P}_\mathsf{r} \mathbf{z}^t_\mathsf{u} \tag{30}$$

The obtained score is then used to modulate the conditional intensity function $(\lambda_{\mathsf{v},\mathsf{r},\mathsf{u}}(t|\mathcal{H}_{t-}))$ of a TPP for a given relation r and entities v and u as follows:

$$\lambda_{\mathsf{v},\mathsf{r},\mathsf{u}}(t|\mathcal{H}_{t-}) = \exp(\mathtt{s}_{\mathsf{v},\mathsf{r},\mathsf{u}}(t))(t - \bar{t}) \tag{31}$$

where $\bar{t}$ represents the most recent time when either v or u was involved in an observation and $t > \bar{t}$. Using $\exp$ ensures that the intensity function is always positive. To predict when relation r will form between v and u, the conditional intensity is converted into conditional density $(\mathtt{f}_{\mathsf{v},\mathsf{r},\mathsf{u}})$ and subsequently an expectation of time over the time horizon is given as output as described in Section 2.7.

Trivedi *et al.* [217] argue that different types of relations evolve at different rates; e.g., liking posts in a social network occurs more frequently than becoming friends. They model the dynamics of the graph by considering two types of relations: 1- *communications* corresponding to node interactions (e.g., liking someone's post in social media), 2- *associations* corresponding to topological evolution (e.g., forming a new friendship). They propose to use different TPPs for these two types of relations. Towards this goal, they assume the embedding function provided by the encoder gives $\mathtt{EMB}(\mathsf{v}) = (\mathbf{z}^t_\mathsf{v})$ and $\mathtt{EMB}(\mathsf{r}) = (\psi_\mathsf{r}, \mathbf{z}^t_\mathsf{r})$ and define the intensity function of their TPP as follows:

$$\lambda_{\mathsf{v},\mathsf{r},\mathsf{u}}(t|\mathcal{H}_{t-}) = \psi_\mathsf{r}\mathtt{log}(1 + \mathtt{exp}(\frac{\mathbf{z}^{t'}_\mathsf{r}[\mathbf{z}^t_\mathsf{v}; \mathbf{z}^t_\mathsf{u}]}{\psi_\mathsf{r}})) \tag{32}$$

where $[\mathbf{z}^t_\mathsf{v}; \mathbf{z}^t_\mathsf{u}]$ is the concatenation of $\mathbf{z}^t_\mathsf{v}$ and $\mathbf{z}^t_\mathsf{u}$. Notice that the above intensity function does not have the $(t-\bar{t})$ term used in Equation (31). Instead, different rates of evolution (ψ_r) for relations of different types are introduced. Zuo *et al.* [261] use the intensity function of a Hawkes process [92, 157]. The intensity of the interaction is obtained by the Euclidean distance between the interacting nodes and an exponentially discounted interaction history of the neighbors.

4.2 Time-Conditioned Decoders

Time-conditioned decoders are decoders whose goal is *not* to predict *when* something will happen. Instead, their goal is to make predictions for specific timestamps given as input. These decoders can be used in two situations: 1- Extrapolation: given a dynamic graph, predict what will happen at a specific time in the future (e.g., predicting who will be the CEO of Apple 2 years from now), 2- Interpolation: given a dynamic graph that contains only a subset of all the temporal observations, predict the missing observations at a specific time in the past (e.g., predicting who has been the CEO of Apple on *2006-04-01*, assuming this piece of information is not explicit in the KG). In other words, time-conditioned decoders predict what happened (or will happen) at some time t where t can be different in different queries. Note that in cases where we want to predict "when" an event happened (or will happen), if the predicted time can only be selected from a small set of (discrete) timestamps, one may still use a time-conditioned decoder. To do so, a prediction is made for each timestamp t and the timestamp having the highest score is returned as the predicted time. In what follows, we describe several existing approaches for dealing with time in such cases.

Dasgupta *et al.* [54] learn an embedding function which, besides taking nodes and relations as input and producing a vector representation, takes timestamp as input as well and outputs a vector representation for it; that is, $\mathtt{EMB}(t) = (\mathbf{z}_t)$. Their encoder is shallow meaning they initialize $\mathbf{z}_t$ for any timestamp t observed in the training data (and other vector representations) randomly and then optimize its values. To make a prediction about whether some edge $(\mathsf{v}, \mathsf{r}, \mathsf{u})$ existed at time t or not, Dasgupta *et al.* [54] first project the node and relation embeddings to the space of t as:

$$\mathbf{z}^t_\mathsf{v} = \mathbf{z}_\mathsf{v} - (\mathbf{z}'_t\mathbf{z}_\mathsf{v})\mathbf{z}_t, \qquad \mathbf{z}^t_\mathsf{r} = \mathbf{z}_\mathsf{r} - (\mathbf{z}'_t\mathbf{z}_\mathsf{r})\mathbf{z}_t, \qquad \mathbf{z}^t_\mathsf{u} = \mathbf{z}_\mathsf{u} - (\mathbf{z}'_t\mathbf{z}_\mathsf{u})\mathbf{z}_t \tag{33}$$

Then they use the TransE dissimilarity score for the projected vectors:

$$||\mathbf{z}_\mathsf{v}^t + \mathbf{z}_\mathsf{r}^t - \mathbf{z}_\mathsf{u}^t|| \tag{34}$$

Leblay and Chekol [133] follow a similar approach as in [54] but instead of projecting the node and relation vectors to the space of t, they add the embedding of t directly in the TransE decoder as:

$$||\mathbf{z}_\mathsf{v} + \mathbf{z}_\mathsf{r} + \mathbf{z}_t - \mathbf{z}_\mathsf{u}|| \tag{35}$$

Similar to the above extensions for TransE, Ma *et al.* [152] develop extensions of some other decoders for static KGs (e.g., DistMult, ComplEx and RESCAL) by adding a timestamp embedding to their score functions. If a shallow encoder is used for the timestamp embeddings (which is the case in the works described so far), then an embedding can only be learned for the timestamps that have been observed in the train set. Therefore, these approaches may not generalize well to the the timestamps not observed in the trian set as a vector representation has not been learned for these timestamps. With the same reason, these models cannot be used effectively for predicting something in a future timestamp as the training data does not contain any future timestamps. Moreover, these models require many parameters and are prone to overfitting when the number of different timestamps in the training data is large.

The model proposed by García-Durán *et al.* [77] addresses the above issues. Instead of relying on an encoder that maps each timestamp to a vector representation, García-Durán *et al.* [77] assume the encoder provides an embedding for each character in the timestamp. Then, to make a prediction about whether some edge $(\mathsf{v}, \mathsf{r}, \mathsf{u})$ existed at time t or not, they sequentially feed $\mathbf{z}_\mathsf{r}$ and the vector representations for each character in t into an LSTM[2] that outputs $\mathbf{z}_\mathsf{r}^t$, a *time-aware* vector representation for relation r at time t. Then, having the vector representations $\mathbf{z}_\mathsf{v}$, $\mathbf{z}_\mathsf{r}^t$, and $\mathbf{z}_\mathsf{u}$, they use one of the static KG decoders (e.g., TransE or DistMult) to compute a score.

Making Predictions for a single timestamp: In cases where all predictions are to be made for a single timestamp or a single time interval, (e.g., predicting what happens in the next snapshot of a DTDG, or predicting what happens in near future *without* predicting when it will happen), the existing approaches mostly use a static decoder from Section 3.1. A notable exception is the work of Zhou *et al.* [257] for link prediction in DTDGs where a point process based on triadic closure is employed. Let v_i, v_j, and v_k be three nodes in a graph at snapshot t. v_i, v_j, and v_k form a *closed triad* if all of them are pair-wise connected, and they form an *open triad* if all but one pair of the nodes are connected to each other. In an open triad, the node that is connected to the other two nodes is called the *center node* of the triad. A fundamental mechanism in the formation and evolution of dynamic networks known as *triad closure* is the process of closed triads being created from open triads [46, 102].

Example 8. Consider the graph in Figure 1(b). In this graph, v_1, v_2 and v_3 form a closed triad. v_2, v_3 and v_4 form an open triad with v_2 being the center node of this open triad.

For two nodes v and u, let $\mathbf{z}_\mathsf{v}^t$ and $\mathbf{z}_\mathsf{u}^t$ represent the embedding of the two nodes at the t^{th} snapshot respectively. Zhou *et al.* [257] model the probability of v and u forming an edge in the next snapshot to be proportional to the number of open triads this edge will close and the similarities of $\mathbf{z}_\mathsf{v}^t$ and $\mathbf{z}_\mathsf{u}^t$ to the embeddings of the center nodes in the open triads involving v and u.

4.3 Staleness

Consider an encoder for a CTDG that updates the node embedding for node v whenever a new observation involving v is made (e.g., when a new edge is added between v and some other node). Assume the last time the encoder updated the embedding for v was at time t_v and currently we are at time t $(> t_\mathsf{v})$. To make a prediction about v at t, many existing decoders use the embedding of v from its last update. However, depending on how long it has passed since t_v (corresponding to $t - t_\mathsf{v}$), the embedding for v may be staled.

To handle the staleness of representations, Kumar *et al.* [128] propose a method to learn how the representation for a node v evolves when no observation involving v (or involving a node that affects v) is made. Let $\texttt{EMB}(\mathsf{v}) = (\mathbf{z}_\mathsf{v})$ and let $\Delta t_\mathsf{v} = t - t_\mathsf{v}$. Following the approach proposed in [14], Kumar

[2] In cases where there are different time modifiers (e.g., *OccurredAt*, *Since*, and *Until*), the encoder provides a vector representation for the time modifier as well and this representation is also fed into the LSTM.

et al. [128] first create a vector representation $\mathbf{z}_{\Delta t_{\mathsf{v}}} \in \mathbb{R}^d$ for Δt_{v} where the i^{th} element of the vector is computed as follows:

$$\mathbf{z}_{\Delta t_{\mathsf{v}}}[i] = \mathbf{w}[i]\Delta t_{\mathsf{v}} + \mathbf{b}[i] \tag{36}$$

where $\mathbf{w}$ and $\mathbf{b}$ are vectors with learnable parameters. Then they compute a new vector representation $\mathbf{z}^t_{\mathsf{v}}$ for v at time t as follows:

$$\mathbf{z}^t_{\mathsf{v}} = (\mathbf{1} + \mathbf{z}_{\Delta t_{\mathsf{v}}}) \odot \mathbf{z}_{\mathsf{v}} \tag{37}$$

where $\mathbf{1} \in \mathbb{R}^d$ is a vector of ones. Having computed $\mathbf{z}^t_{\mathsf{v}}$, instead of using the (potentially) staled representation $\mathbf{z}_{\mathsf{v}}$, Kumar *et al.* [128] use $\mathbf{z}^t_{\mathsf{v}}$ to make predictions about v at time t.

5 Encoders for Dynamic Graphs

In Section 4, we described how the embedding function produced by an encoder can be consumed by a decoder to make predictions. In this section, we describe several general categories of encoders for dynamic graphs.

5.1 Aggregating Temporal Observations

A simple approach for dealing with the temporal aspect of a dynamic graph is through collapsing the dynamic graph into a static graph by aggregating the temporal observations (or the adjacency matrices) over time. Once an aggregated static graph is produced, a static encoder can be used to generate an embedding function.

Liben-Nowell and Kleinberg [146] follow a simple aggregation approach for DTDGs by ignoring the timestamps and taking the sum (or union) of the entries of the adjacency matrices across all snapshots. That is, assuming $\mathbf{A}^1, \dots, \mathbf{A}^T$ represent the adjacency matrices for T timestamps, Liben-Nowell and Kleinberg [146] first aggregate these adjacency matrices into a single matrix as follows:

$$\mathbf{A}_{sum}[i][j] = \sum_{t=1}^{T} \mathbf{A}^t[i][j] \tag{38}$$

Then a static decoder can be applied on $\mathbf{A}_{sum}$ to learn an embedding function. Hisano [96] also follows a similar aggregation scheme where he takes the union of the previous k formation and dissolution matrices of a DTDG. He defines the formation (dissolution) matrix for snapshot t as a matrix representing which edges have been added (removed) since $(t-1)^{\text{th}}$ snapshot. These simple approaches lose the timing information and may not perform well when timing information are of high importance.

An alternative to taking a uniform average of the adjacency matrices is to give more weights to snapshots that are more recent [202, 106, 3, 4]. Below is one such aggregation:

$$\mathbf{A}_{wsum}[i][j] = \sum_{t=1}^{T} \theta^{T-t} \mathbf{A}^t[i][j] \tag{39}$$

where $0 \leq \theta \leq 1$ controls the importance of recent snapshots.

Example 9. Let $DTDG = \{\mathcal{G}^1, \mathcal{G}^2, \mathcal{G}^3\}$ be a DTDG with three snapshots. Let all $\mathcal{G}^i$s have the same set $\{\mathsf{v}_1, \mathsf{v}_2, \mathsf{v}_3\}$ of nodes and the adjacency matrices be as follows:

$$\mathbf{A}^1 = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \quad \mathbf{A}^2 = \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix} \quad \mathbf{A}^3 = \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix}$$

The aggregation scheme in Equation (38) and Equation (39) (assuming $\theta = 0.5$) respectively aggregate the three adjacency matrices into $\mathbf{A}_{sum}$ and $\mathbf{A}_{wsum}$ as follows:

$$\mathbf{A}_{sum} = \begin{bmatrix} 0 & 3 & 2 \\ 3 & 0 & 2 \\ 2 & 2 & 0 \end{bmatrix} \qquad \mathbf{A}_{wsum} = \begin{bmatrix} 0 & \frac{7}{4} & \frac{3}{2} \\ \frac{7}{4} & 0 & \frac{3}{4} \\ \frac{3}{2} & \frac{3}{4} & 0 \end{bmatrix}$$

Then an embedding function can be learned using $\mathbf{A}_{sum}$ or $\mathbf{A}_{wsum}$ (e.g., by using decomposition approaches). Although the interaction evolution between v_1 and v_3 (which were not connected at the beginning, but then formed a connection) is quite different from the interaction evolution between v_2 and v_3 (which were connected at the beginning and then got disconnected), $\mathbf{A}_{sum}$ assigns the same number to both these pairs. $\mathbf{A}_{wsum}$ contains more temporal information compared to $\mathbf{A}_{sum}$, but still loses great amounts of information. For instance, it is not possible to realize from $\mathbf{A}_{wsum}$ that v_2 and v_3 got disconnected only recently.

The approaches based on aggregating temporal observations typically enjoy advantages such as simplicity, scalability, and the capability to directly use a large body of literature on learning from static graphs. However, these approaches may lose great amounts of useful information hindering them from making accurate predictions in many scenarios.

5.2 Aggregating Static Features

Rather than first aggregating a dynamic graph over time to produce a static graph and then running static encoders on the aggregated graph, in the case of DTDGs, one may first apply a static encoder to each snapshot and then aggregate the results over time. Let $\mathcal{DTDG} = \{\mathcal{G}^1, \ldots, \mathcal{G}^T\}$ be a DTDG. The main idea behind the approaches in this category is to first use a static encoder (e.g., an encoder from Section 3.2.1) to compute/learn node features $\mathbf{z}^t_\mathsf{v}$ for each node v at each timestamp t. The features for each timestamp are computed/learned independently of the other timestamps. Then, these features are aggregated into a single feature vector that can be fed into a decoder.

Yao *et al.* [242] aggregate features into a single feature vector as follows:

$$\mathbf{z}_\mathsf{v} = \sum_{t=1}^{T} \mathtt{exp}(-\theta(T-t))\mathbf{z}^t_\mathsf{v} \tag{40}$$

thus exponentially decaying older features. Zhu *et al.* [258] follow a similar strategy where they compute features for each pair of nodes and take a weighted sum (with prefixed weights) of the features, giving higher weights to the features coming from more recent snapshots.

Rather than using an explicitly defined aggregator (e.g., exponential decay) that assigns prefixed weights to previous snapshots, one can fit a time-series model to the features from previous snapshots and use this model to predict the values of the features for the next snapshot. For the time-series model, Huang and Lin [101] and [85] use the ARIMA model [20], da Silva Soares and Prudêncio [49] use ARIMA and other models such as moving averages, and Moradabadi and Meybodi [163] use an approach based on some basic reinforcement learning.

Scalability: Depending on the number of snapshots and the static encoder used for feature generation, the approaches that compute node features/embeddings at each snapshot independently of the other snapshots and then aggregate these features may be computationally expensive. In the upcoming subsections, for some static encoders (e.g., for decomposition and random-walk approaches), we will see some ways to save computations in later snapshots by leveraging the computations from the previous snapshots.

5.3 Time as a Regularizer

A common approach to leverage the temporal aspect of DTDGs is to use time as a regularizer to impose a smoothness constraint over the embeddings of each node over consecutive snapshots [31, 43, 120, 86, 242, 259, 257]. Let $\mathcal{DTDG} = \{\mathcal{G}^1, \ldots, \mathcal{G}^T\}$. For a node v, let $\mathtt{EMB}^{t-1}(\mathsf{v}) = (\mathbf{z}^{t-1}_\mathsf{v})$ represent the vector representation learned for this node at the $(t-1)^{\text{th}}$ snapshot. To learn the vector representation for v at the t^{th} snapshot, the approaches in this class typically use a static encoder to learn an embedding function for $\mathcal{G}^t$ with the additional constraint that for all $\mathsf{v} \in \mathcal{V}^t$, $\mathtt{dist}(\mathbf{z}^{\mathtt{t}-1}_\mathsf{v}, \mathbf{z}^{\mathtt{t}}_\mathsf{v})$ should be small. This constraint is often called the *smoothness* constraint. A common choice for the distance function is the Euclidean distance:

$$\mathtt{dist}(\mathbf{z}^t_\mathsf{v}, \mathbf{z}^{t-1}_\mathsf{v}) = ||\mathbf{z}^t_\mathsf{v} - \mathbf{z}^{t-1}_\mathsf{v}|| \tag{41}$$

but depending on the (static) encoder being used, other distance functions may be used (see, e.g., [43]). Singer *et al.* [205] add a rotation projection to align the embedding $\mathbf{z}^t_\mathsf{v}$s with the embedding

$\mathbf{z}_{\mathsf{v}}^{t-1}$s before taking the Euclidean distance. Their distance function can be represented as follows:

$$\mathtt{dist}(\mathbf{z}_{\mathsf{v}}^{t}, \mathbf{z}_{\mathsf{v}}^{t-1}) = ||\mathbf{R}^{t}\mathbf{z}_{\mathsf{v}}^{t} - \mathbf{z}_{\mathsf{v}}^{t-1}|| \tag{42}$$

where $\mathbf{R}^t$ is a rotation matrix. Liu *et al.* [149] also use time as a regularizer, but they turn the representation learning problem into a constrained optimization problem that can be approximated in a reasonable amount of time so their representations can be updated fast as new observations are made and so their model may be used for streaming scenarios. The model they propose also handles addition of new nodes to the graph. Pei *et al.* [181] propose a dynamic factor graph model for node classification in which they use the temporal information in a similar way. They impose factors that decrease the probability of the worlds where the label of a node at the t^{th} snapshot is different from the previous snapshots (exponentially decaying the importance of the labels for the older snapshots).

Imposing smoothness constraints through penalizing the distance between the vector representations of a node at consecutive snapshots stops the vector representation from having sharp changes. While this may be desired for some applications, in some other applications a node may change substantially from one snapshot to the other. As an example, if a company gets acquired by a large company, it is expected that its vector representation in the next snapshot makes sharp changes. Instead of penalizing the distance of the vector representations for a node at consecutive snapshots, one may simply initialize the representations (or the model) for time t with the learned representations (or model) at time $t-1$ and then allow the static encoder to further optimize the representation at time t (see, e.g., [81]). This procedure implicitly imposes the smoothness constraint while also allowing for sharp changes when necessary.

Another notable work where time is used as a regularizer is an extension of a well-known model for static graphs, named *LINE* [214], to DTDGs by Du *et al.* [65]. Besides using time as regularizer, the authors propose a way of recomputing the node embeddings only for the nodes that have been influenced greatly from the last snapshot.

5.4 Decomposition-based Encoders

A good application of decomposition methods to dynamic graphs is to use them as an alternative to aggregating temporal observations described in Section 5.1. Let $\mathcal{DTDG} = \{\mathcal{G}^1, \ldots, \mathcal{G}^T\}$. As was proposed in [66], the adjacency matrices $\mathbf{A}^1, \ldots, \mathbf{A}^T$ for T timestamps can be stacked into an order 3 tensor $\mathcal{A} \in \mathbb{R}^{N \times N \times T}$. Then one can do a d-component tensor decomposition (for example, CP decomposition, see [91]):

$$\mathcal{A} \approx \sum_{k=1}^{d} \lambda_k \mathbf{a}_k \otimes \mathbf{b}_k \otimes \mathbf{c}_k \tag{43}$$

where $\lambda_k \in \mathbb{R}_+$, $\mathbf{a}_k, \mathbf{b}_k \in \mathbb{R}^N$, $\mathbf{c}_k \in \mathbb{R}^T$, and $\otimes$ is a tensor product of vector spaces. The temporal pattern is captured in the $\mathbf{c}_k$s, and a combination of $\mathbf{a}_k$s and $\mathbf{b}_k$s can be used as the node (or edge) embeddings. In particular, Dunlavy *et al.* [66] used the Holt-Winters method (see, [33]): given the input $\mathbf{c}_k$, it predicts an L-dimensional vector $\mathbf{c}'_k$, which is the prediction of the temporal factor for the next L timesteps. Then they predict the adjacency tensor for the next L snapshots as $\hat{\mathcal{A}} = \sum_{k=1}^{d} \lambda_k \mathbf{a}_k \otimes \mathbf{b}_k \otimes \mathbf{c}'_k$. One can also use other forms of tensor decomposition, *e.g.* Tucker decomposition or HOSVD [187]. Xiong *et al.* [236] propose a probabilistic factorization of $\mathcal{A}$ where the nodes are represented as normal distributions with the means coming from $\mathbf{a}_k$s and $\mathbf{b}_k$s. They also impose a smoothness prior over the temporal vectors corresponding to using time as a regularizer (see Section 5.3). After some time steps, one needs to update the tensor decomposition for more accurate future predictions. The recomputation can require too much time, so one can try incremental updates (see [84, 137]).

Yu *et al.* [245] presented another way of incorporating temporal dependencies into the embeddings with decomposition methods. As above, let $\mathbf{A}^1, \ldots, \mathbf{A}^T$ be the adjacency matrices for T timestamps. Yu *et al.* [245] predict $\hat{\mathbf{A}}^{T+l}$, where $l \in \mathbb{N}$, as follows. First, they solve the optimization problem:

$$\min \sum_{t=T-\omega}^{T} e^{-\theta(T-t)} ||\mathbf{A}^t - \mathbf{U}(\mathbf{V}^t)'(\mathbf{P}^t)'||_F^2, \tag{44}$$

where $\mathbf{P}^t = (1-\alpha)(\mathbf{I} - \alpha\sqrt{\mathbf{D}^t}\mathbf{A}^t\sqrt{\mathbf{D}^t})^{-1}$ is the projection onto feature space, which ensures the smoothness property (that the neighboring nodes have similar feature vectors, see [245] for details),

ω is a window of timestamps into consideration, $\alpha \in (0, 1)$ is a regularization parameter, θ is a decay parameter, $\mathbf{U} \in \mathbb{R}^{N\times d}$ is a matrix that does not depend on time, and $\mathbf{V}^t \in \mathbb{R}^{N\times d}$ is a matrix with explicit time dependency (in the paper, it is a polynomial in time with matrix coefficients). The optimization problem can be slightly rewritten using sparsity of $\mathbf{A}$ and then solved with stochastic gradient descent. The prediction can be obtained as $\hat{\mathbf{A}}^{T+l} = \mathbf{U}(\mathbf{V}^{T+l})'$. In the encoder framework, the matrix $\mathbf{U}$ can be interpreted as persistent features of the nodes (here, as above, one takes the ith row of the matrix as an embedding of the ith node); and the matrices $\mathbf{V}^t$ are time-dependent features.

The streaming scenario. As was discussed in subsection 3.2.3, one can learn node embedding using either eigen-decomposition or svd for graph matrices for each timestamp. Then one can aggregate these features as in Section 5.2 for predictions. However, recalculating decomposition every time may be quite expensive computationally. So one needs to come up with incremental algorithms that will update the current state in the streaming case.

Incremental eigenvalue decomposition [36, 141, 228] is based on perturbation theory. Consider a generalized eigenvalue problem as in Equation (18). Then assume that in the next snapshot we add of few new edges to the graph $\mathcal{G}^T$. In this case, the Laplacian and the degree matrix change by a small amount: $\Delta\mathbf{L}$ and $\Delta\mathbf{D}$ respectively. Assume that we have solved Equation (18) and $\{(\lambda_i, \mathbf{y}_i)\}_{i=1}^N$ is the solution. Then one can find the solution to the new generalized eigenvalue problem for the graph $\mathcal{G}^{T+1}$ in the form: updated eigenvalues $\approx \lambda + \Delta\lambda$ and updated eigenvectors $\approx \mathbf{y} + \Delta\mathbf{y}$, where $\Delta\lambda$ and $\Delta\mathbf{y}$ can be efficiently computed. For example,

$$\Delta\lambda_i = \frac{\mathbf{y}_i'\Delta\mathbf{L}\mathbf{y}_i - \lambda_i\mathbf{y}_i'\Delta\mathbf{D}\mathbf{y}_i}{\mathbf{y}_i'\mathbf{D}\mathbf{y}_i}. \tag{45}$$

An analogous formula could be written for $\Delta\mathbf{y}_i$. The Davis-Kahan theorem [55] gives an approximation error for the top d eigen-pairs.

As was shown by Levin *et al.* [139], one can recalculate the adjacency spectral embedding (see section 3.2.3 for the construction) in case of addition of a new node v to a graph $\mathcal{G}$. Denote $\mathbf{a}_{\mathsf{v}} \in \mathbb{R}^N$, a binary vector, where each entry indicates whether there is an edge between the added node and an already existing node. Then one can find $\mathbf{z}_{\mathsf{v}}$ as the solution to the maximum likelihood problem to fit $\mathbf{a}_{\mathsf{v}} \sim \text{Bernoulli}(\mathbf{Z}'\mathbf{z}_{\mathsf{v}})$, where $\mathbf{Z}$ is as in Formula (19).

Brand [21] propose an efficient way to update the singular value decomposition of a matrix $\mathbf{S}$ when another lower rank matrix of the same size $\Delta\mathbf{S}$ is added to it. Consider the problem in Equation (21). If one knows the solution $(\mathbf{U}_s, \mathbf{U}_t, \mathbf{\Sigma})$ and $\Delta\mathbf{S}$ is an update of the matrix, one can find a general formula for the update of the svd using some basic computations with block matrices. However, this becomes especially efficient if we approximate the increment as a rank one matrix: $\Delta\mathbf{S} = \mathbf{a}\mathbf{b}'$ (see also [209]). Bunch and Nielsen [25] also treated the case how to update the svd, if a row or column of matrix $\mathbf{S}$ was added or deleted. This can be applied to get the encoding for $\mathcal{DTDG}$ in the case of node addition or deletion.

One problem with incremental updates is that the approximation error keeps accumulating gradually. As a solution one needs to recalculate the model from time to time. However, since the recalculation is expensive, one needs to find a proper time when the error becomes intolerable. Usually in applications people use heuristic methods (e.g., restart after a certain time), however a timing should depend on the graph dynamics. Zhang *et al.* [253] propose a new method where given a tolerance threshold, it notifies at what timestamp the approximation error exceeds the threshold.

5.5 Random Walk Encoders

Recently, several approaches have been propose to leverage or extend the random walk models for static graphs to dynamic graphs. In this section, we provide an overview of these approaches.

Let $\mathcal{DTDG} = \{\mathcal{G}^1, \ldots, \mathcal{G}^T\}$. Mahdavi *et al.* [154] first generate random walks on $\mathcal{G}^1$ similar to the random walk models on static graphs and then feed those random walks to a model $\mathcal{M}^1$ that learns to produce vector representations for nodes given the random walks. For the tth snapshot, instead of generating random walks from scratch, they keep the *valid* random walks from $(t-1)$th snapshot, where they define a random walk as valid if all its nodes and the edges taken along the walk are still in the graph in the tth snapshot. They generate new random walks *only* starting from the affected nodes, where affected nodes are the nodes that have been either added in this snapshot, or are involved in one or more edge addition or deletion. Having obtained the updated random walks, they initialize $\mathcal{M}^t$ with the learned parameters from $\mathcal{M}^{t-1}$ and then allow $\mathcal{M}^t$ to be optimized and produce the node embeddings for the t^{th} snapshot.

Sajjad *et al.* [195] observed that by keeping the valid random walks from the previous snapshot and naively generating new random walks starting from the affected nodes, the resulting random walks may be biased. That is, the random walks obtained by following this procedure may have a different distribution than generating random walks for the new snapshot from scratch. Example 10 demonstrates one such example.

Example 10. Consider Figure 1(a) as the first snapshot of a DTDG and assume the following random walks have been generated for this graph (two random walks starting from each node) following a uniform transition:

$$1)\ \mathsf{v}_1, \mathsf{v}_2, \mathsf{v}_1 \qquad 2)\ \mathsf{v}_1, \mathsf{v}_2, \mathsf{v}_3 \qquad 3)\ \mathsf{v}_2, \mathsf{v}_1, \mathsf{v}_3$$
$$4)\ \mathsf{v}_2, \mathsf{v}_3, \mathsf{v}_1 \qquad 5)\ \mathsf{v}_3, \mathsf{v}_2, \mathsf{v}_1 \qquad 6)\ \mathsf{v}_3, \mathsf{v}_1, \mathsf{v}_2$$

Now assume the graph in Figure 1(b) represents the next snapshot. The affected nodes are v_2, which has a new edge, and v_4, which has been added in this snapshot. A naive approach for updating the above set of random walks is to remove random walks 3 and 4 (since they start from an affected node) and add two new random walks from v_2 and two from v_4. This may give the following eight walks:

$$1)\ \mathsf{v}_1, \mathsf{v}_2, \mathsf{v}_1 \qquad 2)\ \mathsf{v}_1, \mathsf{v}_2, \mathsf{v}_3 \qquad 3)\ \mathsf{v}_2, \mathsf{v}_4, \mathsf{v}_2 \qquad 4)\ \mathsf{v}_2, \mathsf{v}_3, \mathsf{v}_1$$
$$5)\ \mathsf{v}_3, \mathsf{v}_2, \mathsf{v}_1 \qquad 6)\ \mathsf{v}_3, \mathsf{v}_1, \mathsf{v}_2 \qquad 7)\ \mathsf{v}_4, \mathsf{v}_2, \mathsf{v}_3 \qquad 8)\ \mathsf{v}_4, \mathsf{v}_2, \mathsf{v}_1$$

In the above 8 random walks, the number of times a transition from v_2 to v_4 has been made is 1 and the number of times a transition from v_2 to v_1 (or v_3) has been made is 3, whereas, if new random walks are generated from scratch, the two numbers are expected to be the same. The reason for this bias is that in random walks 1, 2, 5, and 6, the walk could not go from v_2 to v_4 as v_4 did not exist when these walks were generated. Note that performing more random walks from each node does not solve the bias problem.

Sajjad *et al.* [195] propose an algorithm for generating unbiased random walks for a new snapshot while reusing the valid random walks from the previous snapshot. NetWalk [248] follows a similar approach as the previous two approaches. However, rather than relying on NLP techniques to generate vector representations for nodes given random walks, they develop a customized autoencoder model that learns the vector representations for nodes while minimizing the pairwise distance among the nodes in each random walk.

The previous three approaches mainly leverage the temporal aspect of DTDGs to reduce the computations. They can be useful in the case of feature aggregation (see Section 5.2) when random walk encoders are used to learn features at each snapshot. However, they may fail at capturing the evolution and the temporal patterns of the nodes. Nguyen *et al.* [170, 169] propose an extension of the random walk models for CTDGs that also captures the temporal patterns of the nodes.

Let $\mathcal{CTDG} = (\mathcal{G}, \mathcal{O})$ be a continuous-time graph such that the only type of event is the addition of new edges. Therefore, the nodes are fixed and each element of $\mathcal{O}$ can be represented as $(AddEdge, (\mathsf{v}, \mathsf{u}), t_{(\mathsf{v},\mathsf{u})})$ indicating an edge was added between v and u at time $t_{(\mathsf{v},\mathsf{u})}$. Nguyen *et al.* [170, 169] constrained the random walks to respect time, where they define a random walk on $\mathcal{CTDG}$ that respects time as follows.

A random walk on $\mathcal{CTDG}$ that respects time is a sequence $\mathsf{v}_1, \mathsf{v}_2, \ldots, \mathsf{v}_l$ of nodes where:

$$\mathsf{v}_i \in \mathcal{V} \text{ for all } 1 \leq i \leq l \tag{46}$$
$$(AddEdge, (\mathsf{v}_i, \mathsf{v}_{i+1}), t_{(\mathsf{v}_i,\mathsf{v}_{i+1})}) \in \mathcal{O} \text{ for all } 1 \leq i \leq l-1 \tag{47}$$
$$t_{(\mathsf{v}_i,\mathsf{v}_{i+1})} \leq t_{(\mathsf{v}_{i+1},\mathsf{v}_{i+2})} \text{ for all } 1 \leq i \leq l-2 \tag{48}$$

That is, the sequence of edges taken by each random walk only moves forward in time. Similar to the random walks on static graphs, the initial node to start a random walk from and the next node to transition to can come from a distribution. Unlike the static graphs, however, these distributions can be a function of time. For instance, consider a walk that has currently reached a node u by taking an edge (v, u) that has been added at time t. The edge for the next transition (to be selected from the outgoing edges of u that have been added after t) can be selected with a probability proportional to how long after t they were added to the graph.

Example 11. Assume $t_1 < t_2 < t_3 < t_4 < t_5$ for the CTDG in Figure 1(d). Consider a random walk that has started at v_1, then transitioned to v_2, and is now deciding the next node to transition to. According to Nguyen *et al.* [170]'s strategy, even though both v_4 and v_5 are neighbors of v_2, only the transition to v_4 is valid as the edge between v_2 and v_4 has been added after the edge between v_1 and v_2 whereas the edge between v_2 and v_5 has been added before the edge between v_1 and v_2.

De Winter *et al.* [58] follow a similar approach as in [170] but for DTDGs instead of CTDGs. Their experiments show that in some cases, discretizing a CTDG into a DTDG and then running the random walks that respect time on the DTDG results in better performance. Bastas *et al.* [12] also follow a similar approach as in [170], but they divide the time horizon into two intervals one corresponding to observations before some time t_s and the other corresponding to the observations after t_s. They aggregate the observations until time t_s into a static graph (see Section 5.1) following the intuition that older observations mainly contain topological information and not temporal information. They run static random walks on the static graph from the first interval and temporal random walks that respect time (with custom distributions for selecting the initial node and the node to transition to) on the second interval. Both the static and temporal walks are then used to learn node embeddings.

5.5.1 Analysis of Random Walk Encoders

Supervised and unsupervised learning: Similar to decomposition and autoencoder-based approaches, one of the major advantages of the encoders based on random walks is that they provide an embedding function without needing to be combined with a decoder. Therefore, the encoder can be used for unsupervised learning approaches such as clustering and community detection [235, 246]. However, the disconnect between the encoder and the decoder typically prevents these models from being trained end-to-end. Therefore, for supervised prediction tasks, the embedding learned for nodes are not optimized for the prediction problem.

The streaming scenario: When new observations are made, random walk approaches typically require to perform new walks that take the new observations into account and then update the node embeddings based on the new walks. This update usually requires a few rounds of computing gradients, which, depending on the size of the dynamic graph, can be slow. This makes random walk approaches not an ideal option for the streaming scenario.

Random walk for KGs and for attributed graphs: Using random walks for representation learning has been mostly done for static and dynamic graphs. While the idea of random walks has been also used for reasoning over KGs [132, 52], the use of random walks for representation learning on KGs has not yet been explored much. An interesting direction for future research is to extend the approaches discussed in this section to dynamic KGs.

5.6 Sequence-Model Encoders

A natural choice for modeling dynamic graphs is by extending sequence models to graph data. With the success of RNNs in several synchronous sequence modeling problems [160, 9, 94, 158, 103, 200], where the duration between any two consecutive items in the sequence is considered equal, and several asynchronous sequence modeling problems [45, 143, 64, 167, 260, 100], RNNs have been the most common choice for extending sequence models to DTDGs and CTDGs. In the next subsections, we describe the RNN-based models for DTDGs, which can be considered a synchronous sequence modeling problem, and CTDGs, which can be considered an asychronous modeling problem. We also describe some other sequence modeling tools that have been extended to dynamic graphs.

5.6.1 RNN-based Encoders for DTDGs

Let $\mathcal{DTDG} = \{\mathcal{G}^1, \mathcal{G}^2, \ldots, \mathcal{G}^T\}$ be a DTDG. Let $\mathcal{M}$ be a (differentiable) encoder, which, given a static graph $\mathcal{G}^t$, outputs a vector representation for each node. As an example, $\mathcal{M}$ can be a GCN.

One way of leveraging RNNs for DTDGs is as follows. We run $\mathcal{M}$ on each $\mathcal{G}^t$ and obtain a sequence $\mathbf{z}_\mathsf{v}^1, \mathbf{z}_\mathsf{v}^2, \ldots, \mathbf{z}_\mathsf{v}^T$ of vector representations for each node v. This sequence is then fed into an RNN that produces a vector representation $\mathbf{z}_\mathsf{v}$ for v containing information from v's history and evolution. These vector representations of the nodes can then be fed into a decoder to make predictions about the nodes. The idea behind this approach is similar to the static feature aggregation idea described in Section 5.2 except that the weights of the RNN and the model $\mathcal{M}$ are learned simultaneously and over all the snapshots. The t^{th} step of the encoder for this architecture can be represented as:

$$\mathbf{z}^t_{\mathsf{v}_1}, \ldots, \mathbf{z}^t_{\mathsf{v}_N} = \mathcal{M}(\mathcal{G}^t) \tag{49}$$

$$\mathbf{h}^t_{\mathsf{v}_j} = \texttt{RNN}(\mathbf{h}^{t-1}_{\mathsf{v}_j}, \mathbf{z}^t_{\mathsf{v}_j}) \text{ for } j \in [1, N] \tag{50}$$

which can be equivalently represented as:

$$\mathbf{Z}^t = \mathcal{M}(\mathcal{G}^t) \tag{51}$$

$$\mathbf{H}^t = \texttt{RNN}(\mathbf{H}^{t-1}, \mathbf{Z}^t) \tag{52}$$

where $\mathbf{Z}^t \in \mathbb{R}^{N \times d}$ represents the vector representations of size d for the N nodes in the graph at snapshot t and $\mathbf{H}^t \in \mathbb{R}^{N \times d}$ represents the hidden state of the RNN corresponding to vector representations of size d for the N nodes in the graph that captures the history of the nodes as well. In this architecture, $\mathcal{M}$ aims at capturing the structural information for each node at each snapshot and the RNN aims at capturing the temporal information. The approach described above has been proposed and used in different works. Model 1 of [201] uses this approach where $\mathcal{M}$ is the GCN proposed in [59] and the RNN is a standard LSTM. Narayan and Roe [166] also use this approach with $\mathcal{M}$ being the GCN proposed in [174] and the RNN being a standard LSTM. Manessi *et al.* [155] modify this approach slightly by (mainly) adding skip-connections in the GCN part. Another similar architecture is proposed in [162]. Instead of obtaining $\mathbf{z}^t_{\mathsf{v}_j}$s by running a GCN that aggregates the features of neighbouring nodes only at the t^{th} snapshot, Yu *et al.* [249] propose a 3D GCN that aggregates the features of neighbouring nodes on a window of previous snapshots (i.e., the aggregation is both spatial and temporal rather than just being spatial).

In the above approach, $\mathcal{M}$ is independent of the RNN. That is, the vector representations for nodes provided by $\mathcal{M}$ are independent of the node histories captured in $\mathbf{h}^t_{\mathsf{v}_j}$s. The *embedded approaches* aim at embedding the model(s) $\mathcal{M}$ into the RNN so that $\mathcal{M}$ can also use the node histories.

One such embedded approach has been proposed in [39], where the authors combine the GCN proposed in [59] with LSTMs. Let $\mathcal{DTDG} = \{\mathcal{G}^1, \mathcal{G}^2, \ldots, \mathcal{G}^T\}$, let $\mathbf{A}^t$ represent the adjacency matrix for $\mathcal{G}^t$, and let $\mathbf{A}^t[j]$ represent the j^{th} row of $\mathbf{A}^t$ corresponding to the neighborhood of node v_j. Let $\mathbf{C}^{t-1}[j]$ and $\mathbf{H}^{t-1}[j]$ represent the memory and hidden state of the LSTM at time $t-1$ for node v_j. Let $\mathcal{M}_1$, $\mathcal{M}_2$, $\mathcal{M}_3$, $\mathcal{M}_4$ and $\mathcal{M}_5$ be five GCN models (same model with different parameters), where the node representations for $\mathcal{M}_1$, $\mathcal{M}_2$, $\mathcal{M}_4$ and $\mathcal{M}_5$ are initialized according to $\mathbf{H}^{t-1}$ and for $\mathcal{M}_3$ are initialized according to $\mathbf{C}^{t-1}$. Let $\mathcal{M}_i(\mathcal{G})[j]$ represent the vector representation provided by $\mathcal{M}_i$ for node v_j when applied to $\mathcal{G}$. The embedded model of [39] can be formulated as:

$$\mathbf{i}^t = \sigma\left(\mathbf{W}_{ii}\mathbf{A}^t[j] + \mathcal{M}_1(\mathcal{G}^t)[j] + \mathbf{b}_i\right) \tag{53}$$

$$\mathbf{f}^t = \sigma\left(\mathbf{W}_{fi}\mathbf{A}^t[j] + \mathcal{M}_2(\mathcal{G}^t)[j] + \mathbf{b}_f\right) \tag{54}$$

$$\mathbf{C}^t[j] = \mathbf{f}_t \odot \mathcal{M}_3(\mathcal{G}^t)[j] + \mathbf{i}^t \odot \texttt{Tanh}\left(\mathbf{W}_{ci}\mathbf{A}^t[j] + \mathcal{M}_4(\mathcal{G}^t)[j] + \mathbf{b}_c\right) \tag{55}$$

$$\mathbf{o}^t = \sigma\left(\mathbf{W}_{oi}\mathbf{A}^t[j] + \mathcal{M}^5(\mathcal{G}^t)[j] + \mathbf{b}_o\right) \tag{56}$$

$$\mathbf{H}^t[j] = \mathbf{o}^t \odot \texttt{Tanh}\left(\mathbf{C}^t[j]\right) \tag{57}$$

where the above formulation is done for all $1 \leq j \leq N$. The $\mathcal{M}_i$ models are embedded into the LSTM gates and use its memory and hidden state in their computations. The above formulation can be considered as a standard LSTM taking a sequence of adjacency matrices as input, where the gate computations for the memory and hidden states have been replaced with GCN operations to take the graph structure into account. Other similar ways of embedding the $\mathcal{M}$ models into RNNs can be seen in [142, 201]. Instead of embedding a GCN into an RNN, Pareja *et al.* [179] embed an RNN into a GCN by running a GCN with different parameters at each snapshot where the RNN provides the weights of the GCN at the t^{th} snapshot based on the weights of the GCNs at the previous snapshots.

5.6.2 Other Sequence-Model Encoders for DTDGs

Besides RNNs, other sequence models have been also used as encoders for DTDGs. Sarkar *et al.* [198] use Kalman filters to track the embeddings of the nodes through time for a bipartite graph. Each timestep of the Kalman filter corresponds to a snapshot of the DTDG. The t^{th} timestep observes the adjacency matrix $\mathbf{A}^t$ corresponding to the t^{th} snapshot $\mathcal{G}^t$ of the DTDG and updates the node embeddings accordingly. The observations for each element $\mathbf{A}^t$ are considered to be independent of each other and $\mathbf{A}^t[i][j]$ is defined to be proportional to the distance between the embedding vectors of v_i and v_j. To make the computations tractable, this conditional probability of the observation (i.e., the adjacency matrix) given the hidden state (i.e., the embeddings) is approximated by a Gaussian.

Inspired by fully attentive models (see Section 2.6), Sankar *et al.* [196] propose DySAT: a fully attentive model for DTDGs. Let $\mathcal{DTDG} = \{\mathcal{G}^1, \ldots, \mathcal{G}^T\}$. DySAT applies the attention-based GCN model in [223] to each $\mathcal{G}^t$ and obtains $\mathbf{z}_\mathsf{v}^1, \ldots, \mathbf{z}_\mathsf{v}^T$ for every node v, where $\mathbf{z}_\mathsf{v}^t$ encodes the structural information from $\mathcal{G}^t$. Similar to Vaswani *et al.* [222], DySAT adds a positional embedding $\mathbf{p}^t$ to each $\mathbf{z}_\mathsf{v}^t$ and obtains $\hat{\mathbf{z}}_\mathsf{v}^t = \mathbf{z}_\mathsf{v}^t + \mathbf{p}^t$, where each $\mathbf{p}^t$ encodes information about the relative position of the t^{th} snapshot compared to other snapshots. Finally, DySAT applies a multi-head self-attention on $\hat{\mathbf{z}}_\mathsf{v}^1, \ldots, \hat{\mathbf{z}}_\mathsf{v}^T$ to get the final representation of the node to be sent to the decoder.

5.6.3 RNN-based Encoders for CTDGs

RNN-based approaches for CTDGs [128, 216, 151, 217] mainly consider CTDGs where the only possible observation is addition of new edges. They define custom RNNs that update the representations of the source and target nodes forming a new edge (and the representation of the relation between the two nodes in the case of KGs) upon making a new observation $(AddEdge, (\mathsf{v}, \mathsf{u}), t)$ (or $(AddEdge, (\mathsf{v}, \mathsf{r}, \mathsf{u}), t)$ in the case of a KG). One of the main differences in these approaches is in the way they define the embedding function and the way they define their custom RNN.

Kumar *et al.* [128], for instance, consider bipartite graphs and define $\texttt{EMB}(\mathsf{v}) = (\mathbf{z}_\mathsf{v}, \overline{\mathbf{z}}_\mathsf{v})$ for each node v in the graph where $\mathbf{z}_\mathsf{v} \in \mathbb{R}^{d_1}$ and $\overline{\mathbf{z}}_\mathsf{v} \in \mathbb{R}^{d_2}$. The values of $\mathbf{z}_\mathsf{v}$ are optimized directly (similar to shallow encoders), but the values of $\overline{\mathbf{z}}_\mathsf{v}$ are updated using an RNN. Kumar *et al.* [128] consider two different RNNs for updating the source and the target nodes. Upon making a new observation $(AddEdge, (\mathsf{v}, \mathsf{u}), t)$, the two RNNs update $\overline{\mathbf{z}}_\mathsf{v}$ and $\overline{\mathbf{z}}_\mathsf{u}$ as follows:

$$\overline{\mathbf{z}}_\mathsf{v} = \texttt{RNN}_{source}(\overline{\mathbf{z}}_\mathsf{v}, [\overline{\mathbf{z}}_\mathsf{u}; \Delta t_\mathsf{v}; \mathbf{f}]) \tag{58}$$

$$\overline{\mathbf{z}}_\mathsf{u} = \texttt{RNN}_{target}(\overline{\mathbf{z}}_\mathsf{u}, [\overline{\mathbf{z}}_\mathsf{v}; \Delta t_\mathsf{u}; \mathbf{f}]) \tag{59}$$

where Δt_v represents the time elapsed since v's previous interaction (similarly for Δt_u) and $\mathbf{f}$ represents edge features (e.g., it can be the rating a user assigned to a movie). $\texttt{RNN}_{source}$ is a standard RNN that takes as input the current state $\overline{\mathbf{z}}_\mathsf{v}$ and a new input $[\overline{\mathbf{z}}_\mathsf{u}; \Delta t_\mathsf{v}; \mathbf{f}]$, and outputs an updated state for $\overline{\mathbf{z}}_\mathsf{v}$; similarly for $\texttt{RNN}_{target}$. The two vectors $\mathbf{z}_\mathsf{v}$ and $\overline{\mathbf{z}}_\mathsf{v}$ for a node v are then concatenated as one vector and sent to a decoder for making predictions.

Trivedi *et al.* [216] consider KGs and define $\texttt{EMB}(\mathsf{v}) = (\mathbf{z}_\mathsf{v})$ for every node v where $\mathbf{z}_\mathsf{v} \in \mathbb{R}^{d_1}$ and $\texttt{EMB}(\mathsf{r}) = (\mathbf{z}_\mathsf{r})$ for every relation r where $\mathbf{z}_\mathsf{r} \in \mathbb{R}^{d_2}$. They define two custom RNNs that update $\mathbf{z}_\mathsf{v}$ and $\mathbf{z}_\mathsf{u}$ upon making a new observation $(AddEdge, (\mathsf{v}, \mathsf{r}, \mathsf{u}), t)$ as follows:

$$\mathbf{z}_\mathsf{v} = \texttt{Tanh}(\mathbf{W}_s \Delta t_\mathsf{v} + \mathbf{W}_{hh}\texttt{Tanh}(\mathbf{W}_h[\mathbf{z}_\mathsf{v}; \mathbf{z}_\mathsf{u}; \mathbf{r}_{p_\mathsf{v}}])) \tag{60}$$

$$\mathbf{z}_\mathsf{u} = \texttt{Tanh}(\mathbf{W}_t \Delta t_\mathsf{u} + \mathbf{W}_{hh}\texttt{Tanh}(\mathbf{W}_h[\mathbf{z}_\mathsf{u}; \mathbf{z}_\mathsf{v}; \mathbf{r}_{p_\mathsf{u}}])) \tag{61}$$

where Δt_v and Δt_u are defined as before, $\mathbf{r}_{p_\mathsf{v}}$ is the vector representation for the last relation that v was involved in (similarly for $\mathbf{r}_{p_\mathsf{u}}$), and $\mathbf{W}_s \in \mathbb{R}^{d_1 \times 1}$, $\mathbf{W}_t \in \mathbb{R}^{d_1 \times 1}$, $\mathbf{W}_h \in \mathbb{R}^{l \times (2d_1 + d_2)}$, and $\mathbf{W}_{hh} \in \mathbb{R}^{d_1 \times l}$ are weight matrices. The vector representation for relations is optimized directly (similar to shallow encoders).

Trivedi *et al.* [217] develop a model that can be used for several types of graphs. They define $\texttt{EMB}(\mathsf{v}) = (\mathbf{z}_\mathsf{v})$. Upon making a new observation $(AddEdge, (\mathsf{v}, \mathsf{u}, t))$, they update the node representation for v using the following custom RNN (and similarly for u):

$$\mathbf{z}_\mathsf{v} = \phi(\mathbf{W}_1 \mathbf{z}_{\mathcal{N}(\mathsf{u})} + \mathbf{W}_2 \mathbf{z}_\mathsf{v} + \mathbf{W}_3 \Delta t_\mathsf{v}) \tag{62}$$

where $\mathbf{z}_{\mathcal{N}(\mathsf{u})}$ is a weighted aggregation of the neighbours of u, Δt_v is defined as before, $\mathbf{W}_i$s are weight matrices, and ϕ is an activation function. The aggregation $\mathbf{z}_{\mathcal{N}(\mathsf{u})}$ can be different for different types of graphs (e.g., it can take relations into account in the case of a KG). Trivedi *et al.* [217] define a temporal attention mechanism to obtain the neighbor weights for $\mathbf{z}_{\mathcal{N}(\mathsf{u})}$ at each time.

Other ways of defining the embedding function as well as custom RNNs can be viewed in [151] for directed graphs and [50] for bipartite graphs.

5.6.4 Discussion and Analysis of RNN-based Encoders

Information propagation: Upon observing $(AddEdge, (\mathsf{v}, \mathsf{u}), t)$ (or $(AddEdge, (\mathsf{v}, \mathsf{r}, \mathsf{u}), t)$ in the case of a KG), many existing works only update the nodes directly involved in the new edge. Ma *et al.* [151] argue that it is important to propagate this information to the neighboring nodes so that they can update their representations accordingly. Towards this goal, they first compute a vector representation for the new observation as follows:

$$\mathbf{z}_o = \phi(\mathbf{W}_1\mathbf{z}_{s_\mathsf{v}} + \mathbf{W}_2\mathbf{z}_{t_\mathsf{u}} + \mathbf{b}) \quad (63)$$

where $\mathbf{z}_{s_\mathsf{v}}$ and $\mathbf{z}_{t_\mathsf{u}}$ belong to $\mathtt{EMB}(\mathsf{v})$ and $\mathtt{EMB}(\mathsf{u})$ respectively and $\mathbf{W}_1$, $\mathbf{W}_2$ and $\mathbf{b}$ are learnable parameters. $\mathbf{z}_o$ is then sent to the immediate neighbors of v and u and custom RNNs update the representation of the neighbors based on $\mathbf{z}_o$ and based on how they are connected to v and/or u.

Attributed graphs: For attributed graphs where nodes have attributes with fixed values, one way to incorporate these attributes into the model is by initializing (part of) the node representations using their attribute values [217]. For the case where the attribute values can change over time as well, Seo *et al.* [201] and Feng *et al.* [72] develop models that take such changes into account for DTDGs. Developing models for attributed CTDGs where the attribute values can change over time is an interesting direction for future research.

The streaming scenario: For the RNN-based approaches developed for CTDGs, once the RNN weights are learned during training, the RNN has learned how to take an observation as input and update the node (and relation) embeddings without requiring to compute any further gradients. That is, after training, the RNN weights can be freezed and used for updating the representations as new observations arrive. This makes RNN-based approaches a natural choice for the streaming scenario. Although as the amount of data collected during the test (freezed) time increases (e.g., when it reaches some predefined threshold), the training can run again on all the collected data to learn better weights for the RNN, then the weights can be frozen again and the updated RNN can replace the old one.

5.7 Autoencoder-based Encoders

Let $\mathcal{DTDG} = \{\mathcal{G}^1, \ldots, \mathcal{G}^T\}$ and $\mathbf{A}^1, \ldots, \mathbf{A}^T$ be the corresponding adjacency matrices. Goyal *et al.* [81] learn an autoencoder AE^1 for $\mathcal{G}^1$ similar to SDNE where the first component takes as input $\mathbf{A}^1[i]$ and generates a vector representation $\mathbf{z}^1_{\mathsf{v}_i}$ for node v_i. The second component takes $\mathbf{z}^1_{\mathsf{v}_i}$ as input and reconstructs $\mathbf{A}^1[i]$. $\mathbf{z}^1_{\mathsf{v}_i}$ and $\mathbf{z}^1_{\mathsf{v}_j}$ are constrained to be close together if there is an edge between v_i and v_j. Having AE^1, the embedding function for the first snapshot is defined as $\mathtt{EMB}^1(\mathsf{v}_i) = (\mathbf{z}^1_{\mathsf{v}_i})$. For the t^{th} snapshot ($t > 1$), an autoencoder AE^t is initialized with the weights from AE^{t-1} and then trained based on $\mathbf{A}^t$ to produce the vector representations for nodes at snapshot t. AE^t can have a different size (e.g., different number of neurons or layers) compared to AE^{t-1}. The authors decide the size of AE^t based on heuristic methods that take into account the size of AE^{t-1} and how different $\mathcal{G}^t$ is from $\mathcal{G}^{t-1}$. If the size of AE^t is different from AE^{t-1}, in order to still be able to initialize AE^t according to AE^{t-1}, the authors use the Net2WiderNet and Net2DeeperNet approaches from Chen *et al.* [37], which change the number of neurons and the number of layers in an autoencoder without substantially changing the function it computes.

The approach of Goyal *et al.* [81] uses the information within previous snapshots of a DTDG to enable learning an autoencoder for the current snapshot faster. Furthermore, initializing AE^t according to AE^{t-1} implicitly acts as a regularizer imposing a smoothness constraint (see Section 5.3). However, similar to other approaches that leverage the the temporal aspect of a dynamic graphs only for these two benefits, the embeddings learned in their approach may not capture the evolution of the nodes. To better capture the node evolutions, Bonner *et al.* [16] propose to develop autoencoders that reconstruct a node's neighbourhood in the next snapshot(s) given the current snapshot. They use a two-layer GCN as the first component and dot-product as the second component of the autoencoder. The authors also propose a variational autoencoder model where instead of directly learning the embeddings $\mathbf{Z}^t$ in the first component, they learn a Gaussian distribution from which $\mathbf{Z}^t$ is sampled. The Gaussian distribution is parameterized by a mean vector μ and a variance vector γ that are learned using two separate two-layer GCNs with tied parameters on the first layers.

To take more snapshots into account in learning node embeddings, Goyal *et al.* [82] propose to learn a single autoencoder where at snapshot t, the first component of the autoencoder takes as input $\mathbf{A}^{t-l}[i], \mathbf{A}^{t-l+1}[i], \ldots, \mathbf{A}^{t-1}[i]$ and produces a vector $\mathbf{z}^t_{\mathsf{v}_i}$ corresponding to the embedding of v_i at time t, and the second component takes as input $\mathbf{z}^t_{\mathsf{v}_i}$ and reconstructs $\mathbf{A}^t[i]$. Goyal *et al.* [81] propose several ways for modeling the two components of the autoencoder. Examples for the first component include feeding a concatenation $[\mathbf{A}^{t-l}[i]; \mathbf{A}^{t-l+1}[i]; \ldots; \mathbf{A}^{t-1}[i]]$ into a feed-forward neural network or feeding a sequence $\mathbf{A}^{t-l}[i], \mathbf{A}^{t-l+1}[i], \ldots, \mathbf{A}^{t-1}[i]$ into an LSTM. Similar architectures are used for the second component.

Rahman and Al Hasan [190] also follow an autoencoder-based approach by mapping each node-pair (instead of each node) to a hidden representation, which can then be used to predict addition or deletion of edges in the next snapshot. Towards this goal, for each snapshot they compute features for each pair of nodes based on the graphlet transitions (see [185, 186]). Then they concatenate the features for each node-pair from the previous snapshots (similar to [81]) and feed the concatenation to an autoencoder that learns a vector representation for the node-pair.

6 Other Relevant Models and Problems

While we mainly provided an overview of the models conforming to an encoder-decoder framework, there are other active lines of work on modeling dynamic (knowledge) graphs. Here, we briefly review some of the other related works and some similar problems.

6.1 Statistical Relational Models

There is a large body of work under the umbrella of statistical relational learning [189, 125] aiming at marrying logic and probability to build probabilistic relational models. Examples of such models include [191, 57, 121, 116]. These models typically use soft rules such as:

$$< w : \texttt{Friends}(x, y) \wedge \texttt{Friends}(y, z) \Rightarrow \texttt{Friends}(x, z) > \tag{64}$$

where the rule implies "friends of friends are likely to be friends" and the weight w of the rule is a measure of confidence for the rule. A model is created using a combination of such soft rules and predictions are made using logical and probabilistic inference. Different models differ in how they interpret these rules and weights. The rules and the weights may be learned from data.

In comparison to the encoder-decoder framework, statistical relational models naturally capture uncertainty about facts and relations, which is critical in applications where relations are derived from noisy measurements or ambiguous interpretations such as natural language processing. Furthermore, statistical relational models permit joint inference in a principled and interpretable way over the entire graph while taking into account the uncertainty of the facts. However, this comes at a computational price and therefore it is often needed to restrict the expressivity of the model by only using model structures that are known to be tractable [221, 117] or to approximate the inference [220, 119, 24]. In contrast, inference with encoder-decoder models scale linearly with the size of their representation and although the operators used for inference are often questionable, end-to-end learning allows the weights of the operators to be adjusted to yield the best predictions possible for desired tasks.

Statistical relational models have been extended to dynamic cases as well (see, e.g., [76, 194, 178, 68, 105, 35, 34]). In their simplest form, these models can be extended to dynamic cases by adding time as an argument similar to the following soft rule:

$$< w : \texttt{Friends}(x, y, t) \wedge \texttt{Friends}(y, z, t) \wedge \neg\texttt{Friends}(x, z, t) \Rightarrow \texttt{Friends}(x, z, t+1) > \tag{65}$$

where the rule may increase the probability of the worlds where friends of friends become friends in the next snapshot. The amount of increase in the probability is controlled by w. Note how this rule is similar to the triadic closure procedure of Huang *et al.* [102] as it models how closed triads are created from open triads.

Besides the approaches based on soft rules, there exists a family of approaches based on walks where the aim is to learn probabilistic walks on the graphs (or KGs), which, starting from any node v, ends up (probabilistically) at the nodes that have a desired relation with v. These probabilistic walks are different from the random walk approaches discussed in this survey. Examples of such approaches include [131, 132, 52]. These approaches have been shown to be quite similar to the approaches based on soft rules corresponding in some cases to a subset of the soft rule approaches [113].

6.2 Spatiotemporal Graphs

For applications where it is possible to define temporal relations between the nodes, several papers take a DTDG and combine the snapshots through connecting the nodes in different snapshots to get a spatiotemporal graph (st graph), i.e., a graph that spans both space and time (links across different time steps are known as temporal links where as links within a time step are known as spatial links). Then, instead of learning on an evolving graph, a model is learned on the resulting (static) st graph. An st graph can be considered as a KG with spatial and temporal relations.

Structured time series problems such as video activity recognition and segmentation, and traffic flow prediction are examples of applications that have benefited from a graph theoretic formulation by creating st graphs. In the video domain, for instance, a graph is extracted from each video frame and combined with the graphs extracted from other frames. Pandhre *et al.* [177] utilize random walk encoders for st graphs in two ways: 1- creating random walks on the st graph (which snaps both space and time), 2- first creating spatial random walks at the current snapshot and then temporal random walks over the temporal graph obtained by only keeping the edges between nodes in different snapshots. Their two models show superior results compared to several baselines on temporal prediction problems such as trajectory classification.

For activity recognition, Brendel and Todorovic [22] learn a structured activity model from the st graphs obtained from videos. For recognition, they match the st graph of the given video with the per-class learned activity models. Jain *et al.* [109] use RNNs in a structured setting by modeling each node and edge in a st graph using an RNN. To make the learning feasible, they partition the nodes (and edges) using semantic similarities and share the RNNs amongst the nodes (and edges) withing the same partition. Wang and Gupta [225] pose activity recognition as a graph classification problem. They extract two kinds of graphs from a video, namely the similarity and st graph. The similarity graph learns connections between objects that are semantically related to each other whereas the st graph learns connections between objects that overlap in space and time. Then, they utilize graph classification models on the constructed graphs for activity classification. Action recognition has been also modeled as reasoning over a dynamic graph without creating an st graph. Li *et al.* [144] and Ghosh *et al.* [79] use skeleton-based datasets for action recognition. They consider an evolving human skeleton during the course of the action as a DTDG, contrary to above approaches that use heuristics to combine the individual frame-level graphs. Then they perform graph convolutions in both the temporal as well as the spatial domain. While Li *et al.* [144] only use human pose features, Ghosh *et al.* [79] also leverage additional contextual cues such as object features, functional relationships, etc.

Another challenging problem formulated as an st graph is that of traffic flow prediction. Yu *et al.* [247] propose a spatiotemporal convolution block (st ConvBlock) that consists of temporal gated convolutions and spatial gated convolutions. These st ConvBlocks are stacked to obtain feature representations for each node in the graph and for the traffic speed prediction. The traffic flow prediction problem has been also modeled as reasoning over the dynamic graph using RNN-based approaches (as discussed in Section 5.6.1) without creating an st graph (see, e.g., [142, 249]).

6.3 Constructing (Dynamic) KGs from Text

For the case of the KGs, while we focused in this survey on methods for using a (dynamic) KG to make predictions about its past, current, or future state, there is a large body of research on how to construct a (dynamic) KG from text [29, 63, 255, 53]. These approaches typically rely on information extractors to obtain new (probabilistic) facts from text and then add to the KG the new facts that pass a confidence threshold. Besides using the probabilities produced by the information extractors, some works (e.g., [63]) also leverage the predictive models discussed in this survey to obtain a prior probability for the new facts solely based on what is already in the graph (and not based on textual data). A detailed discussion of these approaches is out of the scope of this paper.

7 Applications, Datasets & Codes

In this section, we provide an overview of the main applications of (dynamic) graphs. We describe some of the datasets that are widely used in the community for representation learning for (knowledge) graphs. We also provide links to online code for some of the works we discussed in the paper.

7.1 Applications

Link prediction: A natural problem for a graph is to predict if there is a link (with label r in case of a KG) between two nodes v and u. In the dynamic case, one may be interested in predicting if such a link existed at time t in the past or if it will appear some time in the future. Example applications include KG completion, friend recommendation, and finding biological connections among species. The most common evaluation metrics for this task include: AOC (area under ROC curve), corresponding to the probability that the predictor gives a higher score to a randomly chosen existing link than a randomly chosen nonexistent one, GMAUC, a geometric mean of AOC, and PRAUC (area under precision-recall curve) - see, for example, [41]. Another metric named error rate was considered in Chen *et al.* [41] corresponding to the ratio of the number of mispredicted links to the total number of truly existing links.

Entity/relation prediction: One of the fundamental problems in KGs is to predict missing entities or relations. It is a classical problem where we want to predict either a missing head entity $(?, \mathsf{r}, \mathsf{u})$, a missing tail entity $(\mathsf{v}, \mathsf{r}, ?)$, or a missing relation $(\mathsf{v}, ?, \mathsf{u})$. In the context of a DTDG, one may want to predict the missing entity or relation in the next snapshot. In the case of a CTDG, one may want to predict the missing entity or relation at a specific timestamp t (e.g., $(?, \mathsf{r}, \mathsf{v}, t)$). Leblay and Chekol [133], and Dasgupta *et al.* [54] considered the task of predicting a missing entity in the temporal case. They rank all entities that can potentially be the missing entity and then find the rank of the actual missing entity. They used mean rank (MR), and the percentage of cases where the actual missing entity is ranked among the top K (known as Hits@K) to compare the quality of the results. In addition to the above metrics, García-Durán *et al.* [77] also computed mean reciprocal rank (MRR). MRR is generally reported under two settings: raw and filtered (see [17] for the details).

Recommender systems: The design of dynamic recommender systems is an important applied dynamic graph problem faced by a myriad of e-commerce companies. In an abstract dynamic recommender-system problem, we have a set of users, a set of items, and a set of timestamped interactions between users and items, and we seek to recommend items to users based on their current tastes [234, 128]. As a coarse approximation, we can view a recommender-system problem as one of link prediction (or entity prediction) and attempt to recommend to users the items they are likely to autonomously choose. However, a more fine-grained analysis reveals several complications present in recommender systems that may be absent in other link prediction problems. First, the actual output of a recommender system for a given user is a sequence of slates of recommended items. If the items on an output slate are highly similar, then their utility to the user may be strongly correlated, so that there is a non-trivial risk of the slate being useless to the user. This risk is mitigated with a more diverse slate, even if the diverse slate has lower sum of expected utilities than the uniform slate [112][3]. Second, from the point of view of an e-commerce company, the purpose of a recommender system is typically to maximize profit in the long term. Therefore, it may be desirable to recommend to the user items in which the user has no immediate interest, but which are expected to cause the user to purchase profitable items or click on profitable advertisements. From this perspective, the design of a dynamic recommender system can be seen as a reinforcement learning problem on a dynamic graph.

Time Prediction: For dynamic graphs, besides predicting which event will happen in the future, an interesting problem is to also predict when that event will happen. As compared to other tasks, this task only exists for dynamic networks. For instance, we saw the example in Section 4.1 of predicting when *Bob* will visit *Montreal*. A similar time prediction problem is the temporal scoping problem in KG completion where the goal is to predict missing timestamps (e.g., answering queries such as $(\mathsf{v}, \mathsf{r}, \mathsf{u}, ?)$, where v is known to have had a relation r with u in the past). Sun *et al.* [210] and Trivedi *et al.* [216, 217] used the mean absolute error between the predicted time and ground truth to measure the quality of the results. Dasgupta *et al.* [54] ordered the predicted timestamps in the decreasing order of their probabilities and selected the rank associated with the correct timestamp. They computed the mean rank (MR) to compare the results.

Node classification: Node classification is the problem of classifying graph nodes into different classes. An example of a node classification problem is to predict the political affiliation of the users of a social network based on their attributes, connections, and activities. Node classification is often studied under two settings: transductive and inductive. In the transductive setting (also known as semi-supervised classification), given the labels of a few nodes, we want to predict the labels of the other nodes in the graph. In the inductive setting, the label is to be predicted for new nodes that have not been seen during training. The problem of node classification becomes challenging in the dynamic case as the distribution of the class labels may change over time. Classification accuracy is a

[3]This reasoning resembles Markowitz's portfolio theory [156].

widely used metric for this task. For datasets where the task is a multi-class classification, micro-F1 and macro-F1 scores are also used to measure the performance [48].

Graph classification: Graph classification is the problem of classifying the whole graph into one class from a set of predefined classes. This task can be useful in domains like bioinformatics and social networks. In bioinformatics, for instance, one important application is the protein function classification. For this task where proteins are viewed as graphs. Common graph classification benchmarks include COLLAB [240], PROTEINS [18] and D&D [62]. The performance is typically measured in terms of classification accuracy.

Network clustering: Network clustering or detecting communities in graphs such as social networks, biological networks, etc. is an important problem. A network cluster/community typically refers to a subset of nodes that are densely connected to each other, but loosely connected to the rest of the nodes. One challenge in community detection for dynamic graphs is that one needs to model how communities evolve over time. A common performance measure used for this task is the overlap between the predicted and the true cluster assignments. Xin *et al.* [235] utilize the random walk encoders to find the closely associated nodes for a given node and cluster the global network into overlapping communities. Furthermore, the closely associated nodes are updated when impacted by dynamic events, giving a dynamic community detection method. Chen *et al.* [42] recast community detection as a node-wise classification problem and present a family of graph neural networks for solving the community detection problem in a supervised setting.

Question/query answering: The way search engines respond to our questions has evolved in last few years. While traditionally search engines were aiming at suggesting documents in which the answer to a query question can be potentially found, these days search engines try to directly give the answer to the question. This has become possible in part due to question answering over KGs (QA-KG). There has been interesting research in recent years in this area. In their question answering system, Huang *et al.* [104] considered simple natural language questions that can be answered correctly by predicting the head entity and the relation mentioned in the question. They answer these questions by jointly predicting the question's head entity, relation, and tail entity in the embedded space of a KG and then selecting a fact from the KG that is closest to the predicted embeddings. De Cao *et al.* [56] considered question answering by reasoning over multiple documents. The authors formulated this problem as inference over an entity graph and used GCNs to do multi-step reasoning. Jia *et al.* [110] presented a way to answer temporal questions over existing QA-KG systems. They defined a temporal question as a question that has a temporal expression (date, time, interval, and periodic events) or temporal signal (before, after, during, etc.) in the question or whose answer is temporal. Hamilton *et al.* [89] propose a way of mapping a query formulated as a conjunctive first-order logic formula into a vector representation (corresponding to a query embedding) through geometric operations. The geometric operations leading to query embeddings are jointly optimized with node embeddings such that the query embedding is close in the embedded space to the embeddings of the entities corresponding to the correct answer of the query. For QA-KG, the most popular metrics are precision, recall, F1 score, AUC, and accuracy.

7.2 Datasets

There is a large collection of datasets used for research on static and dynamic graphs in the community. For brevity, we will only survey a representative sample of these datasets. To explore more network datasets, we refer readers to several popular network repositories such as Stanford Large Network Dataset Collection (`https://snap.stanford.edu/data/index.html`), Network Repository (`http://networkrepository.com/index.php`), Social Computing data repository (`http://socialcomputing.asu.edu/pages/datasets`), LINQS (`https://linqs.soe.ucsc.edu/data`), UCI Network Data Repository (`https://networkdata.ics.uci.edu/`), CNetS Data Repository (`http://cnets.indiana.edu/resources/data-repository/`) and Koblenz Network Collection (`http://konect.uni-koblenz.de/networks/`). Section 7 of [48] and Section 7.1 of [251] are also good starting points to explore other datasets.

Table 2 gives a brief summary of the datasets. In Section 7.2.1, we describe some evolving and temporal datasets. In Section 7.2.2, we survey some of the popular traditional datasets that have been widely used in the community, but are not necessarily temporal or dynamic in nature.

[4]`http://snap.stanford.edu/data/#as`

Dataset	Type	Nodes	Edges	Granularity
Social Evolution [153]	Social network	83 [217]	376-791 Associations [217] 2,016,339 Communications [217]	6 mins
Github [217]	Social network	12,328 [217]	70,640-166,565 Associations [217] 604,649 Communications [217]	-
HEP-TH [78]	Citation network	1,424-7,980 [81]	2,556-21,036 [81]	Monthly
Autonomous systems [136]	Communication network	103-6,474[4]	243-13,233	Daily
GDELT [134]	Events knowledge graph	14,018 [216]	31.29M [216]	15 mins
ICEWS [19]	Events knowledge graph	12498 [216]	0.67M [216]	Daily [216]
YAGO [99]	Knowledge graph	15,403 [77]	138,056 [77]	Mostly yearly [77]
Wikidata [133]	Knowledge graph	11,134 [77]	150,079 [77]	Yearly
Reddit [127]	Social network	55,863	858,49	Seconds
Enron [124]	Email network	151 [39]	50.5K [39]	Seconds
FB-Forum [175, 193]	Social network	899 [170]	33.7K [170]	Seconds
Blog	Social network	5,196 [149]	171,743 [149]	-
Cora[5]	Citation network	2708 [149]	5429 [149]	-
Flicker [213]	Social network	1,715,256 [214]	22,613,981 [214]	-
UCI [129]	Communication network	1,899 [149]	59,835 [149]	Seconds
Radoslaw[6] [159]	Email network	167 [170]	82.9K [170]	Seconds
DBLP[7][8]	Citation network	315,159 [248]	743,70 [248]	-
YELP	Bipartite ratings	6,569 [196]	95,361 [196]	Seconds
MovieLens-10M	Bipartite ratings	20,537 [196]	43,760 [196]	Seconds
CONTACT	Face-to-face proximity	274 [39]	28,200 [39]	Seconds
HYPERTEXT09	Face-to-face proximity	113 [39]	20,800 [39]	Seconds

Table 2: A summary of the datasets, the type of data they contain, the number of nodes and edges they contain, and their temporal granularity.

7.2.1 Temporal Datasets

Social Evolution Dataset[9]: The social evolution dataset was released by the MIT Human Dynamics Lab [153] and is used by Trivedi *et al.* [217] for their work. The dataset is collected between Jan 2008 to Sep 2008 and has 83 nodes. As mentioned in Section 4.1, Trivedi *et al.* [217] consider two categories of relations: associations (or topological evolution) and communications (or interactions). The number of associations evolves from 376 initial edges to 791 final edges. The number of communication events (proximity, calls, and SMS) in the dataset is 2,016,339.

Github Dataset: Github is a web-based hosting service for codes. Trivedi *et al.* [217] collected a dataset from Github archives between Jan 2013 and Dec 2013. They consider “following a user” as an associative event and “starring” or “watching” a repository as a communicative event. The dataset has 12,328 nodes. The number of associations evolves from 70,640 initial edges to 166,565 final edges. There are 604,649 communication events between the users in this dataset.

HEP-TH: Gehrke *et al.* [78] created a dataset of arxiv papers in High Energy Physics Theory conference from January 1993 to April 2003. The graph is a citation network where nodes represent papers and directed edges represent the citations. Goyal *et al.* [81] made this an evolving graph by considering all published paper up to that month. Their graph evolves from 1424 nodes to 7980 nodes, and from 2,556 edges to 21,036 edges. Goyal *et al.* [82] and Yu *et al.* [248] also conducted experiments on variants of this dataset.

Autonomous Systems[10]: Autonomous systems graph [136] is a communication network from the Border Gateway Protocol logs. The graph has 733 daily snapshots from Nov 1997 to Jan 2000. The graph grows from 103 to 6,474 nodes and from 243 to 13,233 edges. One unique thing about this dataset is that while most other graphs have only addition of the nodes and edges, this graph

[5]`https://linqs.soe.ucsc.edu/node/236`
[6]`http://networkrepository.com/radoslaw-email.php`
[7]`https://dblp.uni-trier.de/xml`
[8]`https://snap.stanford.edu/data/com-DBLP.html`
[9]`http://realitycommons.media.mit.edu/socialevolution.html`
[10]`http://snap.stanford.edu/data/as-733.html`

has instances of both addition and deletion of nodes and edges. Goyal *et al.* [81] use the first 100 snapshots, and Goyal *et al.* [82] use the last 50 snapshots for their experiments.

GDELT: Global Database of Events, Language, and Tone (GDELT) [134] is an initiative to construct a database of all the events across the globe connecting people, organizations, events, news sources, and locations. Trivedi *et al.* [216] collected a subset of this data from April 1, 2015, to Mar 31, 2016, with the temporal granularity of 15 mins. This subset contains 14,018 nodes, 20 types of relations, and 31.29M edges.

ICEWS[11]**:** Integrated Crisis Early Warning System (ICEWS) dataset [19] contains information about political events with their timestamps. Here the entities represent important political people (presidents, prime ministers, bureaucrats) and countries, and the relations are political scenarios such as negotiate, sign formal agreement, criticize, etc. The data[12] used in [216] is collected from Jan 1, 2014 to Dec 31, 2014 with the temporal granularity of 24 hours and has 12,498 nodes, 260 types of relations, and 0.67M facts. García-Durán *et al.* [77] created two KGs based on the ICEWS dataset. One of these KGs[13] contains information from 2014 and has 6,869 nodes, 230 types of relations, and 96,730 facts. The other one is a longer term KG[14] containing the events occurring between 2005-2015 with 10,094 nodes, 251 types of relations, and 461,329 facts.

YAGO: YAGO[15] [99] is a spatially and temporally enriched version of the Wikipedia knowledge base. The nodes represent people, groups, artifacts and events while the relations represent facts such as wasBornIn, playsFor, isLocatedIn, etc. YAGO contains temporal information in the form of "occursSince" and "occursUntil". The dataset[16] created by García-Durán *et al.* [77] has 15,403 nodes, 34 types of relations and 138,056 facts. Jiang *et al.* [111]'s dataset has 9,513 nodes, 10 types of relations, and 15,914 facts.

Wikidata: Leblay and Chekol [133] created a temporal knowledge base[17] using Wikidata. García-Durán *et al.* [77] considered a subset of this dataset by selecting the most frequent entities along with the relations that include these entities. In their dataset[18], they have 11,134 nodes, 95 types of relations, and 150,079 facts.

Reddit HyperLink Network[19]**:** Subreddit hyperlink network [127] is a directed network extracted from the posts that create hyperlinks from one subreddit to another. Each edge has temporal information, sentiment of the source towards the target, and the text of the source post. The dataset also comes with subreddit embeddings for 51,278 subreddits. There are 55,863 nodes and 858,490 edges in the graph.

Enron[20]**:** Enron email dataset [124] is the network of email exchanges among the employees of Enron. This data was originally released by the Federal Energy Regulatory Commission as part of their investigation. There are several variants [21] [22][23][24] of this dataset available and it has been widely used in the community [170, 58, 39, 196].

FB-FORUM[25]**:**This dataset comes from a Facebook-like online community of students at the University of California, Irvine and was collected in 2004 [175, 193]. This is a bipartite graph where the nodes represents students and groups while the edges represent students' broadcast messages on

[11]`http://www.icews.com/`

[12]`https://github.com/rstriv/Know-Evolve/tree/master/data/icews`

[13]`https://github.com/nle-ml/mmkb/tree/master/TemporalKGs/icews14`

[14]`https://github.com/nle-ml/mmkb/tree/master/TemporalKGs/icews05-15`

[15]`https://www.mpi-inf.mpg.de/departments/databases-and-information-systems/research/yago-naga/yago/`

[16]`https://github.com/nle-ml/mmkb/tree/master/TemporalKGs/yago15k`

[17]`https://staff.aist.go.jp/julien.leblay/datasets/`

[18]`https://github.com/nle-ml/mmkb/tree/master/TemporalKGs/wikidata`

[19]`http://snap.stanford.edu/data/soc-RedditHyperlinks.html`

[20]`https://en.wikipedia.org/wiki/Enron_Corpus`

[21]`https://snap.stanford.edu/data/email-Enron.html`

[22]`https://www.cs.cmu.edu/~enron/`

[23]`http://networkrepository.com/`

[24]`https://www.kaggle.com/wcukierski/enron-email-dataset`

[25]`http://networkrepository.com/fb-forum.php`

the groups. The dataset used by Nguyen *et al.* [170] for their experiments had 899 nodes, and 33.7K edges along with timestamp in Unix time.

UCI[26]**:** This dataset [129] is obtained from the same social network as the one for FB-FORUM. The dataset is a communication network among users along with timestamps. This network has 1,899 nodes and 59,835 edges [151, 196, 248].

YELP[27]**:** The YELP dataset is a subset of YELP's businesses, reviews and user data. It was originally made public as a Kaggle contest. The dataset consists of 6,685,900 reviews by 1,637,138 users for 192,609 businesses. Sankar *et al.* [196] use a part of the YELP dataset where they select the businesses in the state of Arizona and retain businesses that have more than 15 reviews.

MovieLens-10M[28]**:** The MovieLens dataset [90] is a dynamic user-tag interactions dataset. It consists of 10 million ratings and 100,000 tag applications applied to 10,000 movies by 72,000 users. The dataset shows the tagging behavior of users on the movies they rated. Sankar *et al.* [196] utilize a subset of this dataset with 20,537 nodes and 43,760 links.

CONTACT: The CONTACT dataset introduced in [30] is a dynamic network for face-to-face proximity collected through wireless devices carried by people; a link is created between two people whenever they interact. It contains data from 274 people and 28.2K interactions as described in [39].

HYPERTEXT09: The HYPERTEXT09 dataset described in [107] is a proximity network of attendees at the ACM Hypertext 2009 conference. The dataset contains 113 nodes each corresponding to an attendee and 20,800 edges each corresponding to an interaction between two attendees.

7.2.2 Static Datasets

Blog[29]**:** This dataset was collected from the Blog Catalog website. Bloggers follow other bloggers forming graph edges and they categorize their blog under some predefined classes. The graph used by Liu *et al.* [149] has 5,196 nodes, 171,743 edges, and 6 types of node classes. Other works using this dataset include [141, 150].

CiteSeer[30]**:** This is a citation network where papers are considered as nodes and citations are considered as the edges. The broad category of papers are used as the class labels. It has 3,312 nodes, 4,732 edges, and 6 types of node classes. This dataset is widely used including in [149, 123, 145].

Flickr[31] [32]**:** The Flickr dataset [213] is obtained from a network of users on a photo-sharing website. There are class labels that correspond to the groups that users have subscribed to on the website. The instance used by Tang *et al.* [214] contains 1,715,256 nodes and 22,613,981 edges.

7.3 Open-Source Software

There are several open source libraries providing implementations for the papers discussed in the survey. Fey and Lenssen [74] provide an implementation for many GCN-based papers (e.g., [123, 87]) in PyTorch [180]. Wang and Ga [224] also provide implementations for a variety of graph based models. Along with graph structured models, they also make tensor-based models like transformer [222] available, with the intention of facilitating the development of new models combining the two categories. They provide both PyTorch and MXNet [38] backends for this library. Apart from these libraries, the implementation of several techniques covered in the survey is available in independent code repositories. Table 3 provides links to where these implementations can be found.

8 Future Directions & Conclusion

A wide range of real-world problems can be formulated as reasoning over graphs (or networks). Traditionally, graph analytic methods have been mostly focused on static graphs, while in a large

[26]`http://konect.uni-koblenz.de/networks/opsahl-ucsocial`
[27]`https://www.yelp.com/dataset`
[28]`https://grouplens.org/datasets/movielens/10m/`
[29]`http://socialcomputing.asu.edu/datasets/BlogCatalog`
[30]`https://linqs.soe.ucsc.edu/node/236`
[31]`http://socialnetworks.mpi-sws.org/data-imc2007.html`
[32]`https://snap.stanford.edu/data/web-flickr.html`

Reference	Code
Zhou *et al.* [257]	`https://github.com/luckiezhou/DynamicTriad`
Trivedi *et al.* [216]	`https://github.com/rstriv/Know-Evolve`
[253, 21, 257, 81, 82]	`https://github.com/palash1992/DynamicGEM`
Zhang *et al.* [253]	`https://github.com/ZW-ZHANG/TIMERS`
Sajjad *et al.* [195]	`https://github.com/shps/incremental-representation-learning`
Dasgupta *et al.* [54]	`https://github.com/malllabiisc/HyTE`
Kipf and Welling [123]	`https://github.com/tkipf/gcn`
Hamilton *et al.* [87]	`https://github.com/williamleif/GraphSAGE`
Kazemi and Poole [115]	`https://github.com/Mehran-k/SimplE`
Trouillon *et al.* [218]	`https://github.com/ttrouill/complex`
Seo *et al.* [201]	`https://github.com/youngjoo-epfl/gconvRNN`
Chen *et al.* [37]	`https://github.com/soumith/net2net.torch`
Grover and Leskovec [83]	`https://github.com/aditya-grover/node2vec`
Wu *et al.* [234]	`https://github.com/RuidongZ/Recurrent_Recommender_Networks`
Lacroix *et al.* [130]	`https://github.com/facebookresearch/kbc`
Nickel *et al.* [173]	`https://github.com/mnick/holographic-embeddings`
Veličković *et al.* [223]	`https://github.com/PetarV-/GAT`
Liao *et al.* [145]	`https://github.com/lrjconan/LanczosNetwork`
Lerer *et al.* [135]	`https://github.com/facebookresearch/PyTorch-BigGraph`

Table 3: Link to open-source software for static and dynamic (knowledge) graphs.

number of applications the graphs are dynamic and evolve over time. In the past few years, there has been a surge of works on dynamic graphs. In this paper, we surveyed the recent approaches for representation learning over dynamic graphs. We described these approaches according to an encoder-decoder framework, a framework that has gained popularity within several communities.

Our survey sheds light on several ways in which learning from and reasoning with dynamic graphs can be done. Here, we mention some directions to improve learning and reasoning with dynamic graphs.

- Current representation learning algorithms have been mostly designed for discrete-time dynamic graphs (DTDGs), with only a few works on learning from continuous-time dynamic graphs (CTDGs). Even the few existing works for CTDGs are quite limited in the types of observations they can handle as they mainly handle addition of new edges. A promising direction for future research is to extend the existing models for representation learning over CTDGs, or develop new ones, to deal with other types of observations such as edge deletion, node addition, node deletion, node splitting, node merging, etc. Ma *et al.* [150] take some initial steps towards handling node addition, but their proposal provides an embedding for a new node considering only the current state of the graph (not the evolution of the graph).
- While some of the existing encoders work with certain types of graphs, it is not trivial how they can be used for other types of graphs. For instance, random walk approaches have been mainly designed for non-attributed non-labeled graphs. While some initial steps have been taken to extend these approaches to other types of graphs [192], the best way of extending these approaches to the case of attributed graphs and knowledge graphs (KGs) is still not clear. The same goes for autoencoder-based encoders where it is not trivial how these approaches can be extended to KGs, and several other models discussed in the survey. An interesting direction for future research would be to extend these models to be applicable to more types of graphs.
- The approaches for CTDGs that can be used for the streaming scenario are mainly based on RNNs. In other sequence modeling domains (e.g., natural language processing), new sequence modeling approaches have been developed some times showing superior performance compared to RNNs. One example is the transformer architecture [222] where recurrence has been replaced with self-attention. Designing new models for the streaming scenario in CTDGs based on self-attentions is a promising direction for future research. Other possibilities include designing new models based on neural ordinary differential equations (see, e.g., [40]).
- In a CTDG, many observations may be made at the same time. For instance, in an email communications network, a sender may send an email to many receivers at the same time.

Current RNN-based encoders consider a random ordering for such observations. This naive approach may hinder an RNN from learning to generalize to other possible orderings of these simultaneous observations. A future direction would be to extend RNN approaches for dealing with simultaneous observations.

- While there exists a body of research on classifying static graphs and some of them may be (to some extent) applicable to dynamic graphs, the literature on classifying dynamic graphs is still at its infancy. When classification of a dynamic graph is required (e.g., for activity recognition from videos), current approaches often convert the dynamic graph into a static graph and then run a graph classification algorithm on the static graph. Designing dynamic graph classification models is an interesting direction for future research.
- Expressiveness is an important property to be taken into account when selecting/designing a model. A model that is not expressive enough is doomed to underfitting at least for some applications. While a few recent works study the expressiveness of some models for (knowledge) graphs [219, 115, 238, 164], a more detailed and in-depth study of the expressiveness and its empirical impact may be a promising direction for future research.
- The existing models for dynamic graphs only consider edges connecting two nodes in the graph. However, in real-world applications, some edges may connect more than two nodes. These edges are known as hyperedges. In a KG, for instance, an edge corresponding to a *purchase* relation may connect a person as the *buyer* to another person as the *seller* and also to an item that is being purchased. Kazemi [118] argues that representation learning algorithms may fail if these hyperedges are converted into several edges through reifying new entities as during test time, an embedding does not exist for the reified entities. Some recent works study ways of handling such hyperedges for static (knowledge) graphs [233, 73, 239, 10]. A future research direction would be to develop models for dynamic (knowledge) graphs that are capable of handling hyperedges.
- Recently published papers on modeling dynamic graphs are each tested on different datasets, making it difficult to compare these models. It would be quite helpful to create some standard benchmarks with train, validation, and test splits so that future models can be tested on the same benchmarks and splits.

References

[1] Odd Aalen, Ornulf Borgan, and Hakon Gjessing. *Survival and event history analysis: a process point of view*. Springer Science & Business Media, 2008.

[2] Charu Aggarwal and Karthik Subbian. Evolutionary network analysis: A survey. *ACM Computing Surveys (CSUR)*, 47(1):10, 2014.

[3] Nahla Mohamed Ahmed and Ling Chen. An efficient algorithm for link prediction in temporal uncertain social networks. *Information Sciences*, 331:120–136, 2016.

[4] Nahla Mohamed Ahmed, Ling Chen, Yulong Wang, Bin Li, Yun Li, and Wei Liu. Sampling-based algorithm for link prediction in temporal networks. *Information Sciences*, 374:1–14, 2016.

[5] Hirotugu Akaike. Fitting autoregressive models for prediction. *Annals of the institute of Statistical Mathematics*, 21(1):243–247, 1969.

[6] Mohammad Al Hasan and Mohammed J Zaki. A survey of link prediction in social networks. In *Social network Data Analytics*, pages 243–275. Springer, 2011.

[7] László Babai and Ludik Kucera. Canonical labelling of graphs in linear average time. In *20th Annual Symposium on Foundations of Computer Science (sfcs 1979)*, pages 39–46. IEEE, 1979.

[8] Emmanuel Bacry, Iacopo Mastromatteo, and Jean-François Muzy. Hawkes processes in finance. *Market Microstructure and Liquidity*, 1(01):1550005, 2015.

[9] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[10] Song Bai, Feihu Zhang, and Philip HS Torr. Hypergraph convolution and hypergraph attention. *arXiv preprint arXiv:1901.08150*, 2019.

[11] Ivana Balazevic, Carl Allen, and Timothy M Hospedales. Hypernetwork knowledge graph embeddings. *arXiv preprint arXiv:1808.07018*, 2018.

[12] Nikolaos Bastas, Theodoros Semertzidis, Apostolos Axenopoulos, and Petros Daras. evolve2vec: Learning network representations using temporal unfolding. In *International Conference on Multimedia Modeling*, pages 447–458. Springer, 2019.

[13] Mikhail Belkin and Partha Niyogi. Laplacian eigenmaps and spectral techniques for embedding and clustering. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 585–591, 2001.

[14] Alex Beutel, Paul Covington, Sagar Jain, Can Xu, Jia Li, Vince Gatto, and Ed H Chi. Latent cross: Making use of context in recurrent recommender systems. In *Eleventh ACM International Conference on Web Search and Data Mining*, pages 46–54. ACM, 2018.

[15] Cemal Cagatay Bilgin and Bülent Yener. Dynamic network evolution: Models, clustering, anomaly detection. *IEEE Networks*, 2006.

[16] 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.

[17] 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 (NeurIPS)*, pages 2787–2795, 2013.

[18] Karsten M Borgwardt, Cheng Soon Ong, Stefan Schönauer, SVN Vishwanathan, Alex J Smola, and Hans-Peter Kriegel. Protein function prediction via graph kernels. *Bioinformatics*, 21(suppl_1):i47–i56, 2005.

[19] Elizabeth Boschee, Jennifer Lautenschlager, Sean O'Brien, Steve Shellman, James Starz, and Michael Ward. Icews coded event data. *Harvard Dataverse*, 12, 2015.

[20] George EP Box, Gwilym M Jenkins, Gregory C Reinsel, and Greta M Ljung. *Time series analysis: forecasting and control*. John Wiley & Sons, 2015.

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

[22] William Brendel and Sinisa Todorovic. Learning spatiotemporal graphs of human activities. In *Thirteenth International Conference on Computer Vision (ICCV)*, pages 778–785. IEEE, 2011.

[23] Michael M Bronstein, Joan Bruna, Yann LeCun, Arthur Szlam, and Pierre Vandergheynst. Geometric deep learning: going beyond euclidean data. *IEEE Signal Processing Magazine*, 34(4):18–42, 2017.

[24] Hung Hai Bui, Tuyen N Huynh, and Sebastian Riedel. Automorphism groups of graphical models and lifted variational inference. In *Proceedings of the Twenty-Ninth Conference on Uncertainty in Artificial Intelligence (UAI)*, 2013.

[25] J.R. Bunch and C.P. Nielsen. Updating the singular value decomposition. *Numerische Mathematik*, 31:111 – 129, 1978.

[26] Jin-Yi Cai, Martin Fürer, and Neil Immerman. An optimal lower bound on the number of variables for graph identification. *Combinatorica*, 12(4):389–410, 1992.

[27] Hongyun Cai, Vincent W Zheng, and Kevin Chen-Chuan Chang. A comprehensive survey of graph embedding: Problems, techniques, and applications. *IEEE Transactions on Knowledge and Data Engineering*, 30(9):1616–1637, 2018.

[28] Shaosheng Cao, Wei Lu, and Qiongkai Xu. Deep neural networks for learning graph representations. In *Thirtieth AAAI Conference on Artificial Intelligence*, 2016.

[29] Andrew Carlson, Justin Betteridge, Bryan Kisiel, Burr Settles, Estevam R Hruschka, and Tom M Mitchell. Toward an architecture for never-ending language learning. In *Twenty-Fourth AAAI Conference on Artificial Intelligence*, 2010.

[30] Augustin Chaintreau, Pan Hui, Jon Crowcroft, Christophe Diot, Richard Gass, and James Scott. Impact of human mobility on opportunistic forwarding algorithms. *IEEE Transactions on Mobile Computing*, pages 606–620, 2007.

[31] Deepayan Chakrabarti, Ravi Kumar, and Andrew Tomkins. Evolutionary clustering. In *Twelfth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 554–560. ACM, 2006.

[32] Benjamin Paul Chamberlain, James Clough, and Marc Peter Deisenroth. Neural embeddings of graphs in hyperbolic space. *arXiv preprint arXiv:1705.10359*, 2017.

[33] Chris Chatfield and Mohammad Yar. Holt-winters forecasting: Some practical issues. *Journal of the Royal Statistical Society. Series D (The Statistician)*, 37(2), 1988.

[34] Melisachew Wudage Chekol and Heiner Stuckenschmidt. Rule based temporal inference. In *Technical Communications of the 33rd International Conference on Logic Programming (ICLP 2017)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018.

[35] Melisachew Wudage Chekol, Giuseppe Pirrò, Joerg Schoenfisch, and Heiner Stuckenschmidt. Marrying uncertainty and time in knowledge graphs. In *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

[36] Chen Chen and Hanghang Tong. Fast eigen-functions tracking on dynamic graphs. *Society for Industrial and Applied Mathematics Publications*, pages 559–567, 2015.

[37] Tianqi Chen, Ian Goodfellow, and Jonathon Shlens. Net2net: Accelerating learning via knowledge transfer. *arXiv preprint arXiv:1511.05641*, 2015.

[38] Tianqi Chen, Mu Li, Yutian Li, Min Lin, Naiyan Wang, Minjie Wang, Tianjun Xiao, Bing Xu, Chiyuan Zhang, and Zheng Zhang. Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems. *arXiv preprint arXiv:1512.01274*, 2015.

[39] 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.

[40] Tian Qi Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. Neural ordinary differential equations. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 6572–6583, 2018.

[41] Jinyin Chen, Jian Zhang, Xuanheng Xu, Chengbo Fu, Dan Zhang, Qingpeng Zhang, and Qi Xuan. E-lstm-d: A deep learning framework for dynamic network link prediction. *arXiv preprint arXiv:1902.08329*, 2019.

[42] Zhengdao Chen, Lisha Li, and Joan Bruna. Supervised community detection with line graph neural networks. In *International Conference on Learning Representations (ICLR)*, 2019.

[43] Yun Chi, Xiaodan Song, Dengyong Zhou, Koji Hino, and Belle L Tseng. On evolutionary spectral clustering. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 3(4):17, 2009.

[44] 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.

[45] Edward Choi, Mohammad Taha Bahadori, Andy Schuetz, Walter F Stewart, and Jimeng Sun. Doctor AI: Predicting clinical events via recurrent neural networks. In *Machine Learning for Healthcare Conference*, pages 301–318, 2016.

[46] James S Coleman and James Samuel Coleman. *Foundations of social theory*. Harvard University Press, 1994.

[47] David R Cox and Peter Adrian Walter Lewis. Multivariate point processes. In *Sixth Berkeley Symposium on Mathematical Statistics and Probability*, volume 3, pages 401–448, 1972.

[48] Peng Cui, Xiao Wang, Jian Pei, and Wenwu Zhu. A survey on network embedding. *IEEE Transactions on Knowledge and Data Engineering*, 2018.

[49] Paulo Ricardo da Silva Soares and Ricardo Bastos Cavalcante Prudêncio. Time series based link prediction. In *International Joint Conference on Neural Networks (IJCNN)*, pages 1–7. IEEE, 2012.

[50] 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.

[51] Hanjun Dai, Zornitsa Kozareva, Bo Dai, Alex Smola, and Le Song. Learning steady-states of iterative algorithms over graphs. In *International Conference on Machine Learning (ICML)*, pages 1114–1122, 2018.

[52] Rajarshi Das, Shehzaad Dhuliawala, Manzil Zaheer, Luke Vilnis, Ishan Durugkar, Akshay Krishnamurthy, Alex Smola, and Andrew McCallum. Go for a walk and arrive at the answer: Reasoning over paths in knowledge bases using reinforcement learning. In *International Conference on Learning Representations (ICLR)*, 2018.

[53] Rajarshi Das, Tsendsuren Munkhdalai, Xingdi Yuan, Adam Trischler, and Andrew McCallum. Building dynamic knowledge graphs from text using machine reading comprehension. *arXiv preprint arXiv:1810.05682*, 2018.

[54] Shib Sankar Dasgupta, Swayambhu Nath Ray, and Partha Talukdar. Hyte: Hyperplane-based temporally aware knowledge graph embedding. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 2001–2011, 2018.

[55] Chandler Davis and W. M. Kahan. The rotation of eigenvectors by a perturbation. iii. *SIAM Journal of Numerical Analysis*, 7:1–46, 1970.

[56] Nicola De Cao, Wilker Aziz, and Ivan Titov. Question answering by reasoning across documents with graph convolutional networks. *arXiv preprint arXiv:1808.09920*, 2018.

[57] Luc De Raedt, Angelika Kimmig, and Hannu Toivonen. Problog: A probabilistic prolog and its application in link discovery. In *Twentieth International Joint Conference on Artificial Intelligence (IJCAI)*, volume 7, pages 2462–2467. Hyderabad, 2007.

[58] 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 International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*, pages 1234–1241. IEEE, 2018.

[59] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 3844–3852, 2016.

[60] Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Łukasz Kaiser. Universal transformers. *arXiv preprint arXiv:1807.03819*, 2018.

[61] Tim Dettmers, Pasquale Minervini, Pontus Stenetorp, and Sebastian Riedel. Convolutional 2d knowledge graph embeddings. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[62] Paul D Dobson and Andrew J Doig. Distinguishing enzyme structures from non-enzymes without alignments. *Journal of Molecular Biology*, 330(4):771–783, 2003.

[63] Xin Dong, Evgeniy Gabrilovich, Geremy Heitz, Wilko Horn, Ni Lao, Kevin Murphy, Thomas Strohmann, Shaohua Sun, and Wei Zhang. Knowledge vault: A web-scale approach to probabilistic knowledge fusion. In *Twentieth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 601–610. ACM, 2014.

[64] Nan Du, Hanjun Dai, Rakshit Trivedi, Utkarsh Upadhyay, Manuel Gomez-Rodriguez, and Le Song. Recurrent marked temporal point processes: Embedding event history to vector. In *Twenty-Second ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1555–1564. ACM, 2016.

[65] Lun Du, Yun Wang, Guojie Song, Zhicong Lu, and Junshan Wang. Dynamic network embedding: An extended approach for skip-gram based network embedding. In *Twenty-Seventh International Joint Conference on Artificial Intelligence (IJCAI)*, pages 2086–2092, 2018.

[66] 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.

[67] David K Duvenaud, Dougal Maclaurin, Jorge Iparraguirre, Rafael Bombarell, Timothy Hirzel, Alán Aspuru-Guzik, and Ryan P Adams. Convolutional networks on graphs for learning molecular fingerprints. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 2224–2232, 2015.

[68] Maximilian Dylla, Iris Miliaraki, and Martin Theobald. A temporal-probabilistic database model for information extraction. *Very Large Data Bases (VLDB)*, 6(14):1810–1821, 2013.

[69] Carl Eckart and Gale Young. The approximation of one matrix by another of lower rank. *Psychometrika*, 1(211), 1936.

[70] Jeffrey L Elman. Finding structure in time. *Cognitive Science*, 14(2):179–211, 1990.

[71] Robert F Engle and Jeffrey R Russell. Autoregressive conditional duration: a new model for irregularly spaced transaction data. *Econometrica*, pages 1127–1162, 1998.

[72] Fuli Feng, Xiangnan He, Xiang Wang, Cheng Luo, Yiqun Liu, and Tat-Seng Chua. Temporal relational ranking for stock prediction. *arXiv preprint arXiv:1809.09441*, 2018.

[73] Yifan Feng, Haoxuan You, Zizhao Zhang, Rongrong Ji, and Yue Gao. Hypergraph neural networks. *arXiv preprint arXiv:1809.09401*, 2018.

[74] Matthias Fey and Jan E. Lenssen. Fast graph representation learning with PyTorch Geometric. *CoRR*, abs/1903.02428, 2019.

[75] Matthias Fey, Jan Eric Lenssen, Frank Weichert, and Heinrich Müller. Splinecnn: Fast geometric deep learning with continuous b-spline kernels. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 869–877, 2018.

[76] Dov M Gabbay, Christopher John Hogger, and John Alan Robinson. *Handbook of Logic in Artificial Intelligence and Logic Programming: Volume 5: Logic Programming*. Clarendon Press, 1998.

[77] Alberto García-Durán, Sebastijan Dumančić, and Mathias Niepert. Learning sequence encoders for temporal knowledge graph completion. *arXiv preprint arXiv:1809.03202*, 2018.

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

[79] Pallabi Ghosh, Yi Yao, Larry S Davis, and Ajay Divakaran. Stacked spatio-temporal graph convolutional networks for action segmentation. *arXiv preprint arXiv:1811.10575*, 2018.

[80] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. In *International Conference on Machine Learning (ICML)*, pages 1263–1272. JMLR. org, 2017.

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

[82] Palash Goyal, Sujit Rokka Chhetri, and Arquimedes Canedo. dyngraph2vec: Capturing network dynamics using dynamic graph representation learning. *arXiv preprint arXiv:1809.02657*, 2018.

[83] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In *Twnety-Second ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 855–864. ACM, 2016.

[84] Ekta Gujral, Ravdeep Pasricha, and Evangelos E Papalexakis. SamBaTen: Sampling-based batch incremental tensor decomposition. In *SIAM International Conference on Data Mining*, pages 387–395. SIAM, 2018.

[85] İsmail Güneş, Şule Gündüz-Öğüdücü, and Zehra Çataltepe. Link prediction using time series of neighborhood-based node similarity scores. *Data Mining and Knowledge Discovery*, 30(1):147–180, 2016.

[86] Manish Gupta, Charu C Aggarwal, Jiawei Han, and Yizhou Sun. Evolutionary clustering and analysis of bibliographic networks. In *2011 International Conference on Advances in Social Networks Analysis and Mining*, pages 63–70. IEEE, 2011.

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

[88] William L Hamilton, Rex Ying, and Jure Leskovec. Representation learning on graphs: Methods and applications. *IEEE Data Engineering Bulletin*, 40(3):52–74, 2017.

[89] Will Hamilton, Payal Bajaj, Marinka Zitnik, Dan Jurafsky, and Jure Leskovec. Embedding logical queries on knowledge graphs. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 2026–2037, 2018.

[90] F Maxwell Harper and Joseph A Konstan. The movielens datasets: History and context. *ACM Transactions on Interactive Intelligent Systems (TIIS)*, 5(4):19, 2016.

[91] Richard A. Harshman, Peter Ladefoged, H. Graf von Reichenbach, Robert I. Jennrich, Dale Terbeek, Lee Cooper, Andrew L. Comrey, Peter M. Bentler, Jeanne Yamane, and Diane Vaughan. Foundations of the parafac procedure: Models and conditions for an "explanatory" multimodal factor analysis. *UCLA Working Papers Phonetics*, 6:1–84, 1970.

[92] Alan G Hawkes. Spectra of some self-exciting and mutually exciting point processes. *Biometrika*, 58(1):83–90, 1971.

[93] Katsuhiko Hayashi and Masashi Shimbo. On the equivalence of holographic and complex embeddings for link prediction. *arXiv preprint arXiv:1702.05563*, 2017.

[94] Karl Moritz Hermann, Tomas Kocisky, Edward Grefenstette, Lasse Espeholt, Will Kay, Mustafa Suleyman, and Phil Blunsom. Teaching machines to read and comprehend. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 1693–1701, 2015.

[95] Geoffrey E Hinton and Ruslan R Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, 2006.

[96] Ryohei Hisano. Semi-supervised graph embedding approach to dynamic link prediction. In *International Workshop on Complex Networks*, pages 109–121. Springer, 2018.

[97] Frank L Hitchcock. The expression of a tensor or a polyadic as a sum of products. *Journal of Mathematics and Physics*, 6(1-4):164–189, 1927.

[98] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

[99] Johannes Hoffart, Fabian M Suchanek, Klaus Berberich, and Gerhard Weikum. Yago2: A spatially and temporally enhanced knowledge base from wikipedia. *Artificial Intelligence*, 194:28–61, 2013.

[100] Hao Hu and Guo-Jun Qi. State-frequency memory recurrent neural networks. In *International Conference on Machine Learning (ICML)*, pages 1568–1577, 2017.

[101] Zan Huang and Dennis KJ Lin. The time-series link prediction problem with applications in communication surveillance. *INFORMS Journal on Computing*, 21(2):286–303, 2009.

[102] Hong Huang, Jie Tang, Lu Liu, JarDer Luo, and Xiaoming Fu. Triadic closure pattern analysis and prediction in social networks. *IEEE Transactions on Knowledge and Data Engineering*, 27(12):3374–3389, 2015.

[103] Zhiheng Huang, Wei Xu, and Kai Yu. Bidirectional lstm-crf models for sequence tagging. *arXiv preprint arXiv:1508.01991*, 2015.

[104] Xiao Huang, Jingyuan Zhang, Dingcheng Li, and Ping Li. Knowledge graph embedding based question answering. In *Twelfth ACM International Conference on Web Search and Data Mining*, WSDM '19, pages 105–113, New York, NY, USA, 2019. ACM.

[105] Jakob Huber, Christian Meilicke, and Heiner Stuckenschmidt. Applying markov logic for debugging probabilistic temporal knowledge bases. In *Fourth Workshop on Automated Knowledge Base Construction (AKBC)*, 2014.

[106] Nahla Mohamed Ahmed Ibrahim and Ling Chen. Link prediction in dynamic social networks by integrating different types of information. *Applied Intelligence*, 42(4):738–750, 2015.

[107] Lorenzo Isella, Juliette Stehlé, Alain Barrat, Ciro Cattuto, Jean-François Pinton, and Wouter Van den Broeck. What's in a crowd? analysis of face-to-face behavioral networks. *Journal of Theoretical Biology*, 271(1):166–180, 2011.

[108] Valerie Isham and Mark Westcott. A self-correcting point process. *Stochastic Processes and Their Applications*, 8(3):335–347, 1979.

[109] Ashesh Jain, Amir R Zamir, Silvio Savarese, and Ashutosh Saxena. Structural-rnn: Deep learning on spatio-temporal graphs. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 5308–5317, 2016.

[110] Zhen Jia, Abdalghani Abujabal, Rishiraj Saha Roy, Jannik Strötgen, and Gerhard Weikum. Tequila: Temporal question answering over knowledge bases. In *Twenty-Seventh ACM International Conference on Information and Knowledge Management*, pages 1807–1810. ACM, 2018.

[111] Tingsong Jiang, Tianyu Liu, Tao Ge, Lei Sha, Baobao Chang, Sujian Li, and Zhifang Sui. Towards time-aware knowledge graph completion. In *Twenty-Sixth International Conference on Computational Linguistics: Technical Papers*, pages 1715–1724, 2016.

[112] Marius Kaminskas and Derek Bridge. Diversity, serendipity, novelty, and coverage: A survey and empirical analysis of beyond-accuracy objectives in recommender systems. *ACM Transactions on Interactive Intelligent Systems (TiiS)*, 7(1):2, 2017.

[113] Seyed Mehran Kazemi and David Poole. Bridging weighted rules and graph random walks for statistical relational models. *Frontiers in Robotics and AI*, 5:8, 2018.

[114] Seyed Mehran Kazemi and David Poole. RelNN: A deep neural model for relational learning. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[115] Seyed Mehran Kazemi and David Poole. SimplE embedding for link prediction in knowledge graphs. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 4289–4300, 2018.

[116] Seyed Mehran Kazemi, David Buchman, Kristian Kersting, Sriraam Natarajan, and David Poole. Relational logistic regression. In *Fourteenth International Conference on the Principles of Knowledge Representation and Reasoning*, 2014.

[117] Seyed Mehran Kazemi, Angelika Kimmig, Guy Van den Broeck, and David Poole. New liftable classes for first-order probabilistic inference. In *Advances in Neural Information Processing Systems*, pages 3117–3125, 2016.

[118] Seyed Mehran Kazemi. *Representing and learning relations and properties under uncertainty*. PhD thesis, University of British Columbia, 2018.

[119] Kristian Kersting, Babak Ahmadi, and Sriraam Natarajan. Counting belief propagation. In *Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 277–284. AUAI Press, 2009.

[120] Min-Soo Kim and Jiawei Han. A particle-and-density based evolutionary clustering method for dynamic networks. *Very Large Data Bases (VLDB)*, 2(1):622–633, 2009.

[121] Angelika Kimmig, Stephen H Bach, Matthias Broecheler, Bert Huang, and Lise Getoor. A short introduction to probabilistic soft logic. In *NeurIPS Workshop on Probabilistic Programming: Foundations and Applications*, volume 1, page 3, 2012.

[122] John Frank Charles Kingman. P oisson processes. *Encyclopedia of biostatistics*, 6, 2005.

[123] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations (ICLR)*, 2017.

[124] Bryan Klimt and Yiming Yang. Introducing the enron corpus. In *CEAS*, 2004.

[125] Daphne Koller, Nir Friedman, Sašo Džeroski, Charles Sutton, Andrew McCallum, Avi Pfeffer, Pieter Abbeel, Ming-Fai Wong, David Heckerman, Chris Meek, et al. *Introduction to statistical relational learning*. MIT press, 2007.

[126] Jacob Krantz and Jugal Kalita. Abstractive summarization using attentive neural techniques. *arXiv preprint arXiv:1810.08838*, 2018.

[127] Srijan Kumar, William L Hamilton, Jure Leskovec, and Dan Jurafsky. Community interaction and conflict on the web. In *World Wide Web Conference on World Wide Web*, pages 933–943. International World Wide Web Conferences Steering Committee, 2018.

[128] Srijan Kumar, Xikun Zhang, and Jure Leskovec. Learning dynamic embedding from temporal interaction networks. *arXiv preprint arXiv:1812.02289*, 2018.

[129] Jérôme Kunegis. Konect: The koblenz network collection. In *Twenty-Second International Conference on World Wide Web*, WWW '13 Companion, pages 1343–1350, New York, NY, USA, 2013. ACM.

[130] Timothée Lacroix, Nicolas Usunier, and Guillaume Obozinski. Canonical tensor decomposition for knowledge base completion. In *International Conference on Machine Learning (ICML)*, 2018.

[131] Ni Lao and William W Cohen. Relational retrieval using a combination of path-constrained random walks. *Machine Learning*, 81(1):53–67, 2010.

[132] Ni Lao, Tom Mitchell, and William W Cohen. Random walk inference and learning in a large scale knowledge base. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 529–539. Association for Computational Linguistics, 2011.

[133] Julien Leblay and Melisachew Wudage Chekol. Deriving validity time in knowledge graph. In *Companion of the The Web Conference 2018 on The Web Conference 2018*, pages 1771–1776. International World Wide Web Conferences Steering Committee, 2018.

[134] Kalev Leetaru and Philip A Schrodt. Gdelt: Global data on events, location, and tone, 1979–2012. In *ISA Annual Convention*, volume 2, pages 1–49. Citeseer, 2013.

[135] Adam Lerer, Ledell Wu, Jiajun Shen, Timothee Lacroix, Luca Wehrstedt, Abhijit Bose, and Alex Peysakhovich. Pytorch-biggraph: Alarge-scale graph embedding system. *arXiv preprint arXiv:1903.12287*, 2019.

[136] Jure Leskovec, Jon Kleinberg, and Christos Faloutsos. Graphs over time: densification laws, shrinking diameters and possible explanations. In *Eleventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 177–187. ACM, 2005.

[137] Pierre-David Letourneau, Muthu Baskaran, Tom Henretty, James Ezick, and Richard Lethin. Computationally efficient cp tensor decomposition update framework for emerging component discovery in streaming data. In *2018 IEEE High Performance extreme Computing Conference (HPEC)*, pages 1–8, 09 2018.

[138] Ron Levie, Federico Monti, Xavier Bresson, and Michael M Bronstein. Cayleynets: Graph convolutional neural networks with complex rational spectral filters. *IEEE Transactions on Signal Processing*, 67(1):97–109, 2017.

[139] Keith Levin, Farbod Roosta-Khorasani, Michael W Mahoney, and Carey E Priebe. Out-of-sample extension of graph adjacency spectral embedding. *arXiv preprint arXiv:1802.06307*, 2018.

[140] Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard Zemel. Gated graph sequence neural networks. *arXiv preprint arXiv:1511.05493*, 2015.

[141] Jundong Li, Harsh Dani, Xia Hu, Jiliang Tang, Yi Chang, and Huan Liu. Attributed network embedding for learning in a dynamic environment. In *Conference on Information and Knowledge Management (CIKM)*, pages 387–396. ACM, 2017.

[142] Yaguang Li, Rose Yu, Cyrus Shahabi, and Yan Liu. Diffusion convolutional recurrent neural network: Data-driven traffic forecasting. *arXiv preprint arXiv:1707.01926*, 2017.

[143] Yang Li, Nan Du, and Samy Bengio. Time-dependent representation for neural event sequence prediction. *arXiv preprint arXiv:1708.00065*, 2017.

[144] Chaolong Li, Zhen Cui, Wenming Zheng, Chunyan Xu, and Jian Yang. Spatio-temporal graph convolution for skeleton based action recognition. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[145] Renjie Liao, Zhizhen Zhao, Raquel Urtasun, and Richard S Zemel. Lanczosnet: Multi-scale deep graph convolutional networks. *arXiv preprint arXiv:1901.01484*, 2019.

[146] David Liben-Nowell and Jon Kleinberg. The link-prediction problem for social networks. *Journal of the American Society for Information Science and Technology*, 58(7):1019–1031, 2007.

[147] Yankai Lin, Zhiyuan Liu, Maosong Sun, Yang Liu, and Xuan Zhu. Learning entity and relation embeddings for knowledge graph completion. In *Twenty-ninth AAAI conference on artificial intelligence*, 2015.

[148] Hanxiao Liu, Yuexin Wu, and Yiming Yang. Analogical inference for multi-relational embeddings. In *International Conference on Machine Learning (ICML)*, pages 2168–2178. JMLR. org, 2017.

[149] 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 (WWW '19 Companion)*, 2019.

[150] Jianxin Ma, Peng Cui, and Wenwu Zhu. Depthlgp: learning embeddings of out-of-sample nodes in dynamic networks. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[151] Yao Ma, Ziyi Guo, Zhaochun Ren, Eric Zhao, Jiliang Tang, and Dawei Yin. Streaming graph neural networks. *arXiv preprint arXiv:1810.10627*, 2018.

[152] Yunpu Ma, Volker Tresp, and Erik A Daxberger. Embedding models for episodic knowledge graphs. *Journal of Web Semantics*, 2018.

[153] Anmol Madan, Manuel Cebrian, Sai Moturu, Katayoun Farrahi, et al. Sensing the" health state" of a community. *IEEE Pervasive Computing*, 11(4):36–45, 2012.

[154] 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.

[155] Franco Manessi, Alessandro Rozza, and Mario Manzo. Dynamic graph convolutional networks. *arXiv preprint arXiv:1704.06199*, 2017.

[156] Harry Markowitz. Portfolio selection. *The journal of finance*, 7(1):77–91, 1952.

[157] Hongyuan Mei and Jason M Eisner. The neural hawkes process: A neurally self-modulating multivariate point process. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 6754–6764, 2017.

[158] Grégoire Mesnil, Yann Dauphin, Kaisheng Yao, Yoshua Bengio, Li Deng, Dilek Hakkani-Tur, Xiaodong He, Larry Heck, Gokhan Tur, Dong Yu, et al. Using recurrent neural networks for slot filling in spoken language understanding. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 23(3):530–539, 2015.

[159] Radosław Michalski, Sebastian Palus, and Przemysław Kazienko. Matching organizational structure and social network extracted from email communication. In *Lecture Notes in Business Information Processing*, volume 87, pages 197–206. Springer Berlin Heidelberg, 2011.

[160] Tomáš Mikolov, Martin Karafiát, Lukáš Burget, Jan Černockỳ, and Sanjeev Khudanpur. Recurrent neural network based language model. In *Eleventh Annual Conference of the International Speech Communication Association*, 2010.

[161] 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 (NeurIPS)*, pages 3111–3119, 2013.

[162] Sudatta Mohanty and Alexey Pozdnukhov. Graph cnn+ lstm framework for dynamic macroscopic traffic congestion prediction. In *International Workshop on Mining and Learning with Graphs*, 2018.

[163] Behnaz Moradabadi and Mohammad Reza Meybodi. A novel time series link prediction method: Learning automata approach. *Physica A*, 2017.

[164] Christopher Morris, Martin Ritzert, Matthias Fey, William L Hamilton, Jan Eric Lenssen, Gaurav Rattan, and Martin Grohe. Weisfeiler and leman go neural: Higher-order graph neural networks. In *Thirty-Third AAAI Conference on Artificial Intelligence*, 2019.

[165] Kevin Patrick Murphy and Stuart Russell. Dynamic bayesian networks: representation, inference and learning. 2002.

[166] Apurva Narayan and Peter HO'N Roe. Learning graph dynamics using deep neural networks. *IFAC-PapersOnLine*, 51(2):433–438, 2018.

[167] Daniel Neil, Michael Pfeiffer, and Shih-Chii Liu. Phased lstm: Accelerating recurrent network training for long or event-based sequences. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 3882–3890, 2016.

[168] Dat Quoc Nguyen, Kairit Sirts, Lizhen Qu, and Mark Johnson. Stranse: a novel embedding model of entities and relationships in knowledge bases. In *NAACL-HLT*, 2016.

[169] Giang H Nguyen, John Boaz Lee, Ryan A Rossi, Nesreen K Ahmed, Eunyee Koh, and Sungchul Kim. Dynamic network embeddings: From random walks to temporal random walks. In *2018 IEEE International Conference on Big Data (Big Data)*, pages 1085–1092. IEEE, 2018.

[170] Giang Hoang Nguyen, John Boaz Lee, Ryan A Rossi, Nesreen K Ahmed, Eunyee Koh, and Sungchul Kim. Continuous-time dynamic network embeddings. In *Companion of the The Web Conference 2018 on The Web Conference 2018*, pages 969–976. International World Wide Web Conferences Steering Committee, 2018.

[171] Maximilian Nickel, Volker Tresp, and Hans-Peter Kriegel. A three-way model for collective learning on multi-relational data. In *International Conference on Machine Learning (ICML)*, volume 11, pages 809–816, 2011.

[172] Maximilian Nickel, Kevin Murphy, Volker Tresp, and Evgeniy Gabrilovich. A review of relational machine learning for knowledge graphs. *IEEE*, 104(1):11–33, 2016.

[173] Maximilian Nickel, Lorenzo Rosasco, and Tomaso Poggio. Holographic embeddings of knowledge graphs. In *Thirtieth AAAI conference on artificial intelligence*, 2016.

[174] Mathias Niepert, Mohamed Ahmed, and Konstantin Kutzkov. Learning convolutional neural networks for graphs. In *International Conference on Machine Learning (ICML)*, pages 2014–2023, 2016.

[175] T. Opsahl. Triadic closure in two-mode networks: Redefining the global and local clustering coefficients. *Social Networks*, 2011.

[176] Mingdong Ou, Peng Cui, Jian Pei, Ziwei Zhang, and Wenwu Zhu. Asymmetric transitivity preserving graph embedding. In *Twenty-Second ACM SIGKDD Conderence on Knowledge Discovery and Data Mining*, 2016.

[177] Supriya Pandhre, Himangi Mittal, Manish Gupta, and Vineeth N Balasubramanian. Stwalk: learning trajectory representations in temporal graphs. In *ACM India Joint International Conference on Data Science and Management of Data*, pages 210–219. ACM, 2018.

[178] Tivadar Papai, Henry Kautz, and Daniel Stefankovic. Slice normalized dynamic markov logic networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 1907–1915, 2012.

[179] Aldo Pareja, Giacomo Domeniconi, Jie Chen, Tengfei Ma, Toyotaro Suzumura, Hiroki Kanezashi, Tim Kaler, and Charles E Leisersen. Evolvegcn: Evolving graph convolutional networks for dynamic graphs. *arXiv preprint arXiv:1902.10191*, 2019.

[180] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. In *NeurIPS 2017 Autodiff Workshop: The Future of Gradient-based Machine Learning Software and Techniques*, 2017.

[181] Yulong Pei, Jianpeng Zhang, GH Fletcher, and Mykola Pechenizkiy. Node classification in dynamic social networks. *Proceedings of AALTD*, page 54, 2016.

[182] Jeffrey Pennington, Richard Socher, and Christopher Manning. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014.

[183] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: Online learning of social representations. In *Twentieth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 701–710. ACM, 2014.

[184] Bryan Perozzi, Vivek Kulkarni, and Steven Skiena. Walklets: Multiscale graph embeddings for interpretable network classification. *arXiv preprint arXiv:1605.02115*, 2016.

[185] Nataša Pržulj, Derek G Corneil, and Igor Jurisica. Modeling interactome: scale-free or geometric? *Bioinformatics*, 20(18):3508–3515, 2004.

[186] Nataša Pržulj. Biological network comparison using graphlet degree distribution. *Bioinformatics*, 23(2):e177–e183, 2007.

[187] Stephan Rabanser, Oleksandr Shchur, and Stephan Günnemann. Introduction to tensor decompositions and their applications in machine learning. *arXiv preprint arXiv:1711.10781*, 2017.

[188] Lawrence R Rabiner and Biing-Hwang Juang. An introduction to hidden markov models. *ieee assp magazine*, 3(1):4–16, 1986.

[189] Luc De Raedt, Kristian Kersting, Sriraam Natarajan, and David Poole. Statistical relational artificial intelligence: Logic, probability, and computation. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 10(2):1–189, 2016.

[190] 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.

[191] Matthew Richardson and Pedro Domingos. Markov logic networks. *Machine Learning*, 62(1-2):107–136, 2006.

[192] Petar Ristoski and Heiko Paulheim. Rdf2vec: Rdf graph embeddings for data mining. In *International Semantic Web Conference*, pages 498–514. Springer, 2016.

[193] Ryan A. Rossi and Nesreen K. Ahmed. The network data repository with interactive graph analytics and visualization. In *Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.

[194] Adam Sadilek and Henry Kautz. Recognizing multi-agent activities from gps data. In *Twenty-Fourth AAAI Conference on Artificial Intelligence*, 2010.

[195] Hooman Peiro Sajjad, Andrew Docherty, and Yuriy Tyshetskiy. Efficient representation learning using random walks for dynamic graphs. *arXiv preprint arXiv:1901.01346*, 2019.

[196] 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.

[197] Adam Santoro, David Raposo, David G Barrett, Mateusz Malinowski, Razvan Pascanu, Peter Battaglia, and Timothy Lillicrap. A simple neural network module for relational reasoning. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 4967–4976, 2017.

[198] Purnamrita Sarkar, Sajid M Siddiqi, and Geogrey J Gordon. A latent space approach to dynamic embedding of co-occurrence data. In *Artificial Intelligence and Statistics*, pages 420–427, 2007.

[199] Michael Schlichtkrull, Thomas N Kipf, Peter Bloem, Rianne Van Den Berg, Ivan Titov, and Max Welling. Modeling relational data with graph convolutional networks. In *European Semantic Web Conference*, pages 593–607. Springer, 2018.

[200] Minjoon Seo, Aniruddha Kembhavi, Ali Farhadi, and Hannaneh Hajishirzi. Bidirectional attention flow for machine comprehension. *arXiv preprint arXiv:1611.01603*, 2016.

[201] Youngjoo Seo, Michaël Defferrard, Pierre Vandergheynst, and Xavier Bresson. Structured sequence modeling with graph convolutional recurrent networks. In *International Conference on Neural Information Processing*, pages 362–373. Springer, 2018.

[202] Umang Sharan and Jennifer Neville. Temporal-relational classifiers for prediction in evolving domains. In *2008 Eighth IEEE International Conference on Data Mining*, pages 540–549. IEEE, 2008.

[203] Peter Shaw, Jakob Uszkoreit, and Ashish Vaswani. Self-attention with relative position representations. In *Annual Conference of the North American Chapter of the Association for Computational Linguistics*, 2018.

[204] Martin Simonovsky and Nikos Komodakis. Dynamic edge-conditioned filters in convolutional neural networks on graphs. In *IEEE conference on computer vision and pattern recognition*, pages 3693–3702, 2017.

[205] Uriel Singer, Ido Guy, and Kira Radinsky. Node embedding over temporal graphs. *arXiv preprint arXiv:1903.08889*, 2019.

[206] Richard Socher, Danqi Chen, Christopher D Manning, and Andrew Ng. Reasoning with neural tensor networks for knowledge base completion. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 926–934, 2013.

[207] Gustav Sourek, Vojtech Aschenbrenner, Filip Zelezny, Steven Schockaert, and Ondrej Kuzelka. Lifted relational neural networks: Efficient learning of latent relational structures. *Journal of Artificial Intelligence Research (JAIR)*, 62:69–100, 2018.

[208] Myra Spiliopoulou. Evolution in social networks: A survey. In *Social Network Data Analytics*, pages 149–175. Springer, 2011.

[209] Peter Stange. On the efficient update of the singular value decomposition. *PAMM*, 8:10827 – 10828, 2008.

[210] Yizhou Sun, Jiawei Han, Charu C Aggarwal, and Nitesh V Chawla. When will it happen?: relationship prediction in heterogeneous information networks. In *Fifth ACM international Conference on Web Search and Data Mining*, pages 663–672. ACM, 2012.

[211] Daniel L. Sussman, Minh Tang, Donniell E. Fishkind, and Carey E. Priebe. A consistent adjacency spectral embedding for stochastic blockmodel graphs. *Journal of the American Statistical Association*, 107(499):1119–1128, 2012.

[212] Charles Sutton, Andrew McCallum, and Khashayar Rohanimanesh. Dynamic conditional random fields: Factorized probabilistic models for labeling and segmenting sequence data. *Journal of Machine Learning Research*, 8(Mar):693–723, 2007.

[213] Lei Tang and Huan Liu. Relational learning via latent social dimensions. In *Fifteenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '09, pages 817–826, New York, NY, USA, 2009. ACM.

[214] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. Line: Large-scale information network embedding. In *Twenty-Fourth International Conference on World Wide Web*, pages 1067–1077. International World Wide Web Conferences Steering Committee, 2015.

[215] Phi Vu Tran. Learning to make predictions on graphs with autoencoders. In *2018 IEEE 5th International Conference on Data Science and Advanced Analytics (DSAA)*, pages 237–245. IEEE, 2018.

[216] Rakshit Trivedi, Hanjun Dai, Yichen Wang, and Le Song. Know-evolve: Deep temporal reasoning for dynamic knowledge graphs. In *International Conference on Machine Learning (ICML)*, pages 3462–3471. JMLR. org, 2017.

[217] Rakshit Trivedi, Mehrdad Farajtabar, Prasenjeet Biswal, and Hongyuan Zha. Dyrep: Learning representations over dynamic graphs. In *International Conference on Learning Representations (ICLR)*, 2019.

[218] Théo Trouillon, Johannes Welbl, Sebastian Riedel, Éric Gaussier, and Guillaume Bouchard. Complex embeddings for simple link prediction. In *International Conference on Machine Learning (ICML)*, pages 2071–2080, 2016.

[219] Théo Trouillon, Christopher R Dance, Éric Gaussier, Johannes Welbl, Sebastian Riedel, and Guillaume Bouchard. Knowledge graph completion via complex tensor factorization. *The Journal of Machine Learning Research*, 18(1):4735–4772, 2017.

[220] Guy Van den Broeck and Adnan Darwiche. On the complexity and approximation of binary evidence in lifted inference. In *Advances in Neural Information Processing Systems*, pages 2868–2876, 2013.

[221] Guy Van den Broeck. On the completeness of first-order knowledge compilation for lifted probabilistic inference. In *Advances in Neural Information Processing Systems*, pages 1386–1394, 2011.

[222] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 5998–6008, 2017.

[223] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. In *International Conference on Learning Representations (ICLR)*, 2018.

[224] Minjie Wang and Quan Ga. Deep graph library, 2018.

[225] Xiaolong Wang and Abhinav Gupta. Videos as space-time region graphs. In *European Conference on Computer Vision (ECCV)*, pages 399–417, 2018.

[226] Daixin Wang, Peng Cui, and Wenwu Zhu. Structural deep network embedding. In *Twenty-Second ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1225–1234. ACM, 2016.

[227] 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.

[228] Tong Wang, Xing-Sheng He, Ming-Yang Zhou, and Zhong-Qian Fu. Link prediction in evolving networks based on popularity of nodes. *Scientific reports*, 2017.

[229] Yanjie Wang, Rainer Gemulla, and Hui Li. On multi-relational link prediction with bilinear models. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[230] Waloddi Weibull et al. A statistical distribution function of wide applicability. *Journal of Applied Mechanics*, 18(3):293–297, 1951.

[231] Boris Weisfeiler and Andrei A Lehman. A reduction of a graph to a canonical form and an algebra arising during this reduction. *Nauchno-Technicheskaya Informatsia*, 2(9):12–16, 1968.

[232] Greg Welch, Gary Bishop, et al. An introduction to the kalman filter. 1995.

[233] Jianfeng Wen, Jianxin Li, Yongyi Mao, Shini Chen, and Richong Zhang. On the representation and embedding of knowledge bases beyond binary relations. In *Twenty-Fifth International Joint Conference on Artificial Intelligence (IJCAI)*, 2016.

[234] Chao-Yuan Wu, Amr Ahmed, Alex Beutel, Alexander J Smola, and How Jing. Recurrent recommender networks. In *Tenth ACM international Conference on Web Search and Data Mining*, pages 495–503. ACM, 2017.

[235] Yu Xin, Zhi-Qiang Xie, and Jing Yang. An adaptive random walk sampling method on dynamic community detection. *Expert Systems with Applications*, 58:10–19, 2016.

[236] Liang Xiong, Xi Chen, Tzu-Kuo Huang, Jeff Schneider, and Jaime G Carbonell. Temporal collaborative filtering with bayesian probabilistic tensor factorization. In *SIAM International Conference on Data Mining*, pages 211–222. SIAM, 2010.

[237] Lizhen Xu, Jason A Duan, and Andrew Whinston. Path to purchase: A mutually exciting point process model for online advertising and conversion. *Management Science*, 60(6):1392–1412, 2014.

[238] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *International Conference on Learning Representations (ICLR)*, 2019.

[239] Naganand Yadati, Madhav Nimishakavi, Prateek Yadav, Anand Louis, and Partha Talukdar. Hypergcn: Hypergraph convolutional networks for semi-supervised classification. *arXiv preprint arXiv:1809.02589*, 2018.

[240] Pinar Yanardag and SVN Vishwanathan. A structural smoothing framework for robust graph comparison. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 2134–2142, 2015.

[241] Bishan Yang, Wen-tau Yih, Xiaodong He, Jianfeng Gao, and Li Deng. Embedding entities and relations for learning and inference in knowledge bases. In *International Conference on Learning Representations (ICLR)*, 2015.

[242] Lin Yao, Luning Wang, Lv Pan, and Kai Yao. Link prediction based on common-neighbors for dynamic social network. *Procedia Computer Science*, pages 82–89, 2016.

[243] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L Hamilton, and Jure Leskovec. Graph convolutional neural networks for web-scale recommender systems. In *Twenty-Fourth ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 974–983. ACM, 2018.

[244] Zhitao Ying, Jiaxuan You, Christopher Morris, Xiang Ren, Will Hamilton, and Jure Leskovec. Hierarchical graph representation learning with differentiable pooling. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 4800–4810, 2018.

[245] Wenchao Yu, Wei Cheng, Charu C Aggarwal, Haifeng Chen, and Wei Wang. Link prediction with spatial and temporal consistency in dynamic networks. In *Twenty-Sixth International Joint Conference on Artificial Intelligence (IJCAI)*, pages 3343–3349, 2017.

[246] Zhiyong Yu, Jijie Chen, Kun Quo, Yuzhong Chen, and Qian Xu. Overlapping community detection based on random walk and seeds extension. In *Twelfth Chinese Conference on Computer Supported Cooperative Work and Social Computing*, pages 18–24. ACM, 2017.

[247] Bing Yu, Haoteng Yin, and Zhanxing Zhu. Spatio-temporal graph convolutional networks: a deep learning framework for traffic forecasting. In *Twenty-Seventh International Joint Conference on Artificial Intelligence (IJCAI)*, pages 3634–3640. AAAI Press, 2018.

[248] 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 *Twenty-Fourth ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 2672–2681. ACM, 2018.

[249] Bing Yu, Mengzhang Li, Jiyong Zhang, and Zhanxing Zhu. 3d graph convolutional networks with temporal graphs: A spatial information free framework for traffic forecasting. *arXiv preprint arXiv:1903.00919*, 2019.

[250] Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Ruslan R Salakhutdinov, and Alexander J Smola. Deep sets. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 3391–3401, 2017.

[251] Daokun Zhang, Jie Yin, Xingquan Zhu, and Chengqi Zhang. Network representation learning: A survey. *IEEE Transactions on Big Data*, 2018.

[252] Jiani Zhang, Xingjian Shi, Junyuan Xie, Hao Ma, Irwin King, and Dit-Yan Yeung. GaAN: Gated attention networks for learning on large and spatiotemporal graphs. In *Uncertainty in Artificial Intelligence*, 2018.

[253] 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.

[254] Jian Zhang. A survey on streaming algorithms for massive graphs. In *Managing and Mining Graph Data*, pages 393–420. Springer, 2010.

[255] Ce Zhang. Deepdive: a data management system for automatic knowledge base construction. *University of Wisconsin-Madison, Madison, Wisconsin*, 2015.

[256] Jie Zhou, Ganqu Cui, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, and Maosong Sun. Graph neural networks: A review of methods and applications. *arXiv preprint arXiv:1812.08434*, 2018.

[257] 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.

[258] Jia Zhu, Qing Xie, and Eun Jung Chin. A hybrid time-series link prediction framework for large social network. In *International Conference on Database and Expert Systems Applications*, pages 345–359. Springer, 2012.

[259] 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.

[260] Yu Zhu, Hao Li, Yikang Liao, Beidou Wang, Ziyu Guan, Haifeng Liu, and Deng Cai. What to do next: Modeling user behaviors by time-lstm. In *Twenty-Sixth International Joint Conference on Artificial Intelligence (IJCAI)*, pages 3602–3608, 2017.

[261] Yuan Zuo, Guannan Liu, Hao Lin, Jia Guo, Xiaoqian Hu, and Junjie Wu. Embedding temporal network via neighborhood formation. In *Twenty-Fourth ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 2857–2866. ACM, 2018.