Temporal Learning for Dynamic Graph

Yue Cai Zhu

School of Computer Science
McGill University
Montréal, Québec, Canada

June 10, 2022

A thesis submitted to McGill University in partial fulfillment of the requirements of the degree of

Master of Computer Science

Abstract

A graph is a data structure to model a complex system of entities connected in a particular relation. These entities are nodes in the graph, and the connections are edges. A dynamic graph is a graph that evolves in its nodes, edges or both. We can use it to analyze networks that are not static, such as social networks, academic citation networks and city traffic networks. The dynamic graph is a widely used data structure in various domains. However, the exploration of machine learning with dynamic graphs is still in its early stage. What are the drivers of a dynamic graph's evolution? How can we learn the temporal information from a dynamic graph's history? How can we determine if we should use dynamic graph learning algorithms to analyze a given graph? These are still open questions that are well worth to explore. In this thesis, I try to address the research questions above by a survey of recently developed supervised dynamic graph learning algorithms and proposing a dynamic graph temporal learning framework. Based on the framework above, I conducted an initial study on measuring the significance of temporal patterns to prepare for the research in predicting performance gain from dynamic graph learning algorithms.

Abrégé

Le graphe est une structure de données pour modéliser un système complexe d'entités qui sont connectées les unes aux autres dans un certain type de relation. Ces entités sont des sommets dans le graphe et les connexions sont des arêtes. Un graphe dynamique est un graphe qui évolue dans le temps, dans ses sommets, ses arêtes ou les deux. On peut l'utilisé pour analyser les réseaux qui ne sont pas statiques, tels que les réseaux sociaux, les réseaux de citations universitaires et le réseau de trafic urbain. Le graphe dynamique est une structure de données largement utilisée dans domaines divers. Cependant, l'exploration de l'apprentissage automatique avec la graphe dynamique est encore à ses débuts. Quels sont les moteurs de l'évolution d'un graphe dynamique? Comment pouvons-nous apprendre les informations temporelles à partir de l'historique d'un graphe dynamique ? Comment déterminer si nous devons utiliser des algorithmes d'apprentissage de graphes dynamique pour analyser un graphe? Ce sont encore des questions ouvertes qui bien méritent à explorer. Dans cette thèse, j'essaie de répondre les questions de recherche susmentionnées par une enquête sur les algorithmes d'apprentissage supervisé de graphes dynamique récemment développés, et proposer un cadre d'apprentissage temporel de graphe dynamique. Basé sur le cadre d'apprentissage temporel mentionné, j'ai fait une étude préalable sur la mesure de la signification des modifs temporels, et me préparer à la

recherche sur la prédiction du gain de performance offert par les algorithmes d'apprentissage de graphes dynamique.

Acknowledgements

During my study at McGill, I received in-depth training on various topics about Artificial Intelligence (AI) and Machine Learning(ML), especially Natural Language Processing, Network Science and the foundation theory of AI and ML. I appreciate our faculty's effort in developing such a fantastic program syllabus. When I worked with my supervisor Dr.Xue Liu, I learned from him how to do research from a high-level point of view, how to discover and focus on the high-impact research ideas, how to plan our work more efficiently, and many other best practices in conducting scientific research. All of these are very beneficial to my future career.

The collaboration system he established between his students and other research groups also opened up my eyesight to different scientific domains. I am genuinely thankful to my supervisor Dr.Xue Liu.

If there is one thing I regret during my study at McGill, that would be my giving up on the research idea that Dr.Kieran O'Donnell helped me establish. I selected to work on a start-up idea instead of continuing that potential high-impact work. I must say sorry to Dr.Kieran O'Donnell for not continuing to work with him because of my greediness. I also need to say 'thank you' to him for all the favours he did for me. This regretful experience taught me that I should not be greedy for money, reputation, etc. The only reason we

do research is our joyfulness in exploring something that's not been discovered before, the joyfulness we had when we realized we just did some work to help science move forwards.

At last, I need to say thank you to Dr.Peter C. Rigby. He led me into the computer science research community when I studied at Concordia University for my Bachelor's degree. He taught me a lot of software engineering skills that are still helping me in my current position as a machine learning engineer and researcher. He helped me finish and publish my first full research paper as an undergraduate student. He helped me realize my passion for doing scientific research. Without Dr.Peter C. Rigby's guidance, I would be on a completely different route in my life.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Introduction to Graph and Dynamic Graph

Our world is connected. Every object in our world is connected to other entities with a particular relation, such as gravity, energy transmission, etc. We can model a system of connected objects by the data structure *Graph*. A graph is a set of nodes with some of them connected by a specific relation named edges. If all nodes are of the same type, then the resulting graph is called *Homogeneous Graph*. Social graphs, traffic graphs, and academic citation graphs are homogeneous graphs.

We have been studying and applying the data structure graph for a long time to analyze different phenomena presented in our world. It has been discovered that some of the graphs that arise from nature and human social activity evolve. These evolving graphs inspire the idea of *Dynamic Graph* [57]. A dynamic graph is a graph that evolves in its nodes, edges or both. Temporal information revealed by its evolution is the key factor that distinguishes dynamic graphs from static graphs, the kind of graphs that do not change over time.

1.2 The Demand for Supervised Dynamic Graph Learning

Machine learning algorithms for analyzing dynamic graphs can be classified as supervised and unsupervised. Unsupervised dynamic graph learning algorithms do not need the ground truth as labels in their training. Therefore, they are more general than the supervised ones and can be applied even if there is insufficient data. However, they are less accurate than supervised ones. Supervised dynamic graph learning algorithms require ground-truth labels in training. When given enough training data, supervised algorithms can be very accurate.

As the industry collects more and more data, The demand for supervised dynamic graph learning algorithms becomes significant. More and more works exploring supervised dynamic graph learning algorithms emerge in different application domains [72, 65, 106, 102]. The industry is looking for accurate solutions to its graph learning projects. Supervised dynamic graph learning algorithms have a lot of potential to meet their requirements.

1.3 Research Problems and the Thesis Organization

As discussed in the previous subsection, supervised dynamic graph learning algorithms are one potential solution to the industry's need for accurate machine learning models for their graph learning projects. However, the exploration of dynamic graph learning is still in its early stage. There needs to be more understanding of how temporal information is learned in dynamic graph learning. Moreover, there needs to be a general paradigm of how a dynamic graph algorithm should be structured. To address these requirements, Manuscript one surveyed supervised dynamic graph learning algorithms in Chapter 2 and proposed a temporal learning framework. My contributions to manuscript one include its topic selection,

the development of the mentioned temporal learning framework, the categorization and the review of different surveyed algorithms. Manuscript one has been submitted to the journal 'IEEE Transactions on Neural Networks and Learning Systems and is currently in the peer review process.

In chapter 3, according to the temporal learning framework proposed in manuscript one, I developed the Normalized Jaccard Similarity to evaluate the significance of temporal patterns presented in a Discrete Time Dynamic Graph(DTDG) and validate it by experiment. I also discussed my future work in the next phase.

At last, I summarize the work I have done for my learning in the master's program in chapter 4.

Chapter 2

(Manuscript 1)Intelligent Encoder-Decoder Architecture for Dynamic Graph: A Survey

Yue Cai Zhu, Fuyuan Lyu, Chengming Hu, Xi Chen, Xue Liu

Abstract

In recent years, the prevalent online services have generated a sheer volume of user activity data. Service providers collect these data to perform client behaviour analysis and offer better and more customized services. The majority of these data can be modelled and stored as a graph, such as the social graph on Facebook, user-video interaction graph on Youtube. These graphs need to evolve over time to capture the dynamics in the real world, leading to the invention of dynamic graphs. However, the temporal information embedded in the dynamic graphs brings new challenges in analyzing and deploying them. Events staleness, temporal information learning and explicit time dimension usage are some example challenges in dynamic graph learning.

To offer a convenient reference to both the industry and academia, this survey presents the Three Stages Recurrent Temporal Learning Framework based on dynamic graph evolution theories so as to interpret the learning of temporal information with a generalized framework. Under this framework, this survey categorizes and reviews different intelligent encoder-decoder architectures for supervised dynamic graph learning. We believe this survey could supply useful guidelines to researchers and engineers in finding suitable graph structures for their dynamic learning tasks.

2.1 Introduction

In the data explosion era, the amount of data increases exponentially. Most of the data can be viewed as a graph. A graph is a data structure that consists of nodes and edges. It is designed to model and store data that contains not only features for different entities but also relations between them. Graph analysis has long been an important research topic. Previous works [26, 88, 97] assume that the underlying graph is a static graph which does not change over time. But in the real world, the entities modelled as a graph present different temporal dynamics in node features and relations. The dynamic graph is developed to model and store such an evolving graph. The extra time dimension brings temporal information to the graph's representation and reveals the causality embedded in its network dynamic [57]. However, such temporal information also increases the difficulty of analyzing graphs.

In recent years, utilizing machine learning techniques to analyze dynamic graphs has become an emerging research topic [74, 99, 86]. Moreover, the prevalent online services generate a sheer volume of relational data, transaction data and interaction data. They are modelled and stored as attributed dynamic graphs, such as the social graph on Facebook [96] and user video interaction graph on Youtube [9]. Those dynamic graph databases with rich attributes make supervised dynamic graph learning feasible and urge the industry to look for effective supervised dynamic graph learning methods [72, 86, 53] . Therefore, we believe a survey of such methods is extremely helpful for the industry and the research community to exploit the potential of those databases.

There are multiple well-written surveys to summarize dynamic graph representation learning algorithms. Kazemi et al. [43] focus on a broad topic of dynamic graph representation learning. Skarding et al. [79] specialize in Graph Neural Network models for dynamic graphs. They all follow the encoder-decoder learning framework proposed by

Hamilton et al. [32]. With this framework, the encoder generates graph embedding at the node level, and the decoder uses the embedding to perform prediction/classification. Practitioners could assemble different encoder and decoder combinations to best fit their machine learning task. Moreover, the decoder can be modified to perform all dynamic graph learning tasks introduced in later sections. However, the aforementioned surveys do not focus on supervised learning methods, and they do not discuss how temporal information is learnt.

Our work is different from the previous ones mainly in that we develop the Three Stages Recurrent Temporal Learning Framework based on dynamic graph evolution theories. We use this framework to explain how dynamic graphs evolve over time and how different algorithms can learn temporal information. It also gives a general form of these algorithms.

In the development of the mentioned framework, we found that using time as an input feature enables learning algorithms to recognize temporal periodicity and vector clock [51]. Vector clock is recently introduced to describe the phenomena that message sent from neighbouring nodes to the target node requires different traversing time depending on their connection pattern. This motivates us to categorize different algorithms by whether time is learnt implicitly or explicitly, namely Implicit Time and Explicit Time Learning Algorithm. Only Explicit Time learning Algorithms are capable to perform time prediction tasks which predict when a given graph updating event would happen. Time prediction task is recently recognized as one of the goals of dynamic graph learning. [86, 12]

In summary, we make the following contributions in this survey paper:

- The Three Stages Recurrent Temporal Learning Framework for dynamic graph learning. Under this framework, we discuss how temporal information is learnt by different dynamic graph learning algorithms.

- A list of different goals of dynamic graph learning, which includes time prediction.
- A review of the recent development of supervised dynamic graph learning for the Discrete-Time Dynamic Graph and the Continuous-Time Dynamic Graph.
- Some interesting future research directions in dynamic graph learning according to the topics discussed.

The organization of this survey is as follows: Sec. 2.2 introduces some background knowledge of machine learning in dynamic graphs; Sec. 2.3 describes the taxonomy in this survey, mainly on the categories of dynamic graphs, the encoder-decoder learning framework and the motivation of implicit/explicit time learning models categorization. Fig. 2.3 illustrates the taxonomy in this survey; in Sec. 2.4, we introduce the Three Stages Recurrent Temporal Learning Framework . With this framework, we discuss how temporal information is learnt; We will then start the review from algorithms designed for *Discrete Time Dynamic Graph (DTDG)* at Sec. 2.5, and then the algorithms designed for *Continuous Time Dynamic Graph (CTDG)* at Sec. 2.6. Potential future directions are discussed in Sec. 2.7. In Appendix, table 2.6 summarizes the notations and abbreviations used in this work.

2.2 Machine Learning In Dynamic Graph

2.2.1 Supervised Learning in Graph

Supervised machine learning typically trains a machine learning model with historical data and conducts prediction or classification with the trained model during inference time. In order to perform supervised training, the ground truth must be available in the historical data, which is referred to as a label by convention. Supervised learning in graphs differs from

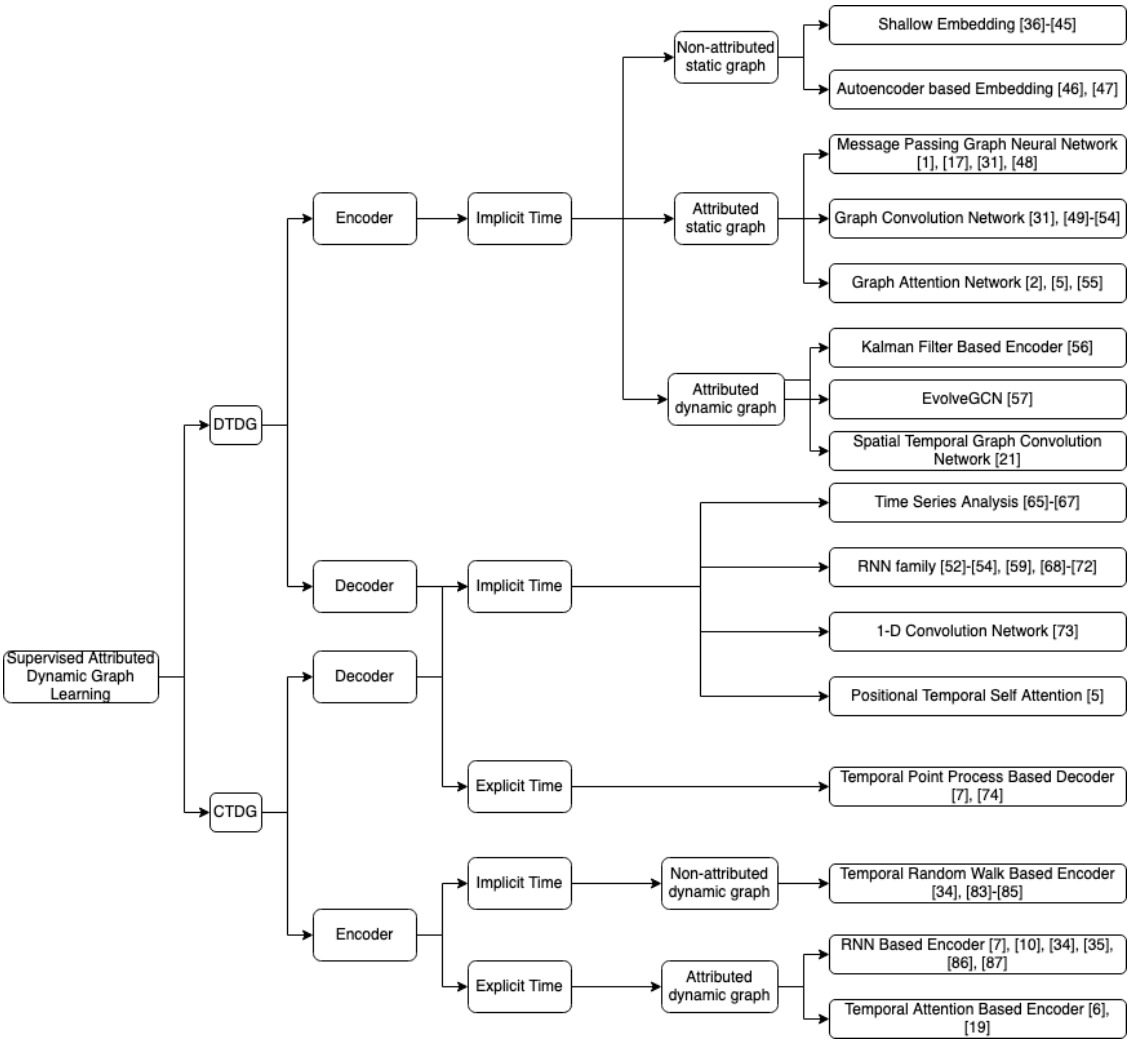


Figure 2.1: Taxonomy

traditional machine learning in that it could be further categorized as node focus task, edge focus task and graph focus task [76]. In the following section, we use classification tasks as examples. Such a paradigm can be easily extended to regression tasks.

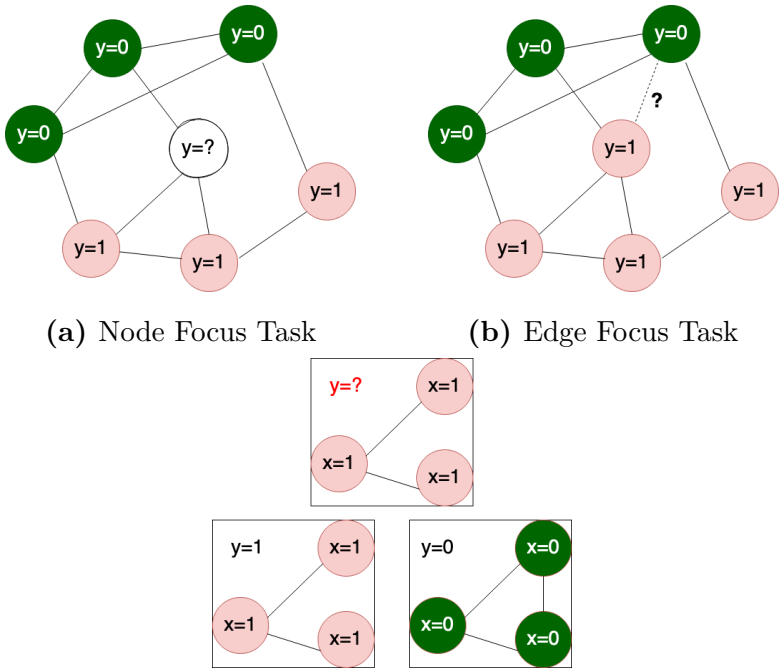
We can denote a graph as $G = (V, E, \mathbf{X})$ with $V = \{v_1, v_2, \ldots, v_i, \ldots, v_m\}$ as the set of nodes in G where $v_i \in G \wedge i \in [1, |V|]$, and $E = \{e_{i,j}\}$ as the set of edges in G where $e_{i,j} = (v_i, v_j, f_{i,j})$, $v_i, v_j \in V$ and $f_{i,j}$ represents the feature of edge $e_{i,j}$. X is the node feature matrix with each row vector $x_{v_i} \in X$ stores the features for node $v_i \in V$. Namely, there exists a one-to-one mapping between V and the row vectors in X. $\mathbf{X} \in \mathcal{R}^{|V| \times d}$ and d is the number of features for a given node.

Then these three tasks can be defined as:

***Definition* 1.** *Node Focus Task: given a one to one mapping between* $V_{known} = \{v_1, v_2, \cdots, v_i\}$ *with* $V_{known} \subset V$ *and the label set* $Y_{known} = \{y_1, y_2, \cdots, y_i\}$. *The learning purpose is to predict* y_k *for* $v_k \in V_{unknown}$ *where* $V_{unknown} \cap V_{known} = \phi$, $V_{unknown} \subseteq V - V_{known}$.

Fig. 2.2a illustrates the idea of the node focus task. The colour represents the node attribute. And each node has its label y. The node focus task is to predict the unknown node labels. As an example of a node focus task, in community detection, the label assigned to each node would be the name of its associated community. If two nodes have the same label, they belong to the same community. It is also straightforward to see that node classification is a node focus task.

***Definition* 2.** *Edge Focus Task: given a one to one mapping between* $E_{known} \subset E$ *and their corresponding labels* Y_{known}. *The learning purpose is to predict* $y_{i,j}$ *for* $e_{i,j} \in E_{unknown}$ *where* $E_{unknown} = E - E_{known}$



(a) Node Focus Task **(b)** Edge Focus Task

(c) Graph Focus Task

Figure 2.2: Different Graph Learning Tasks

As shown in Fig. 2.2b, given a graph with node attribute and observed edges, the edge focus task is to predict whether an edge exists between two given nodes.

***Definition* 3.** *Graph Focus Task: given a collection of graphs or sub-graphs* $\{G_i\}$ *and a one to one mapping between* $\{G_i\}$ *and the label set* $\{y_i\}$*, the learning purpose is to predict* y_j *for* $G_j \notin \{G_i\}$*.*

Given multiple graphs, as in Fig. 2.2c, some of them have known labels, but some do not. The graph focus task is to predict the unknown graph label.

To sum up, the label y to be predicted could be a cell in the adjacency matrix A or an edge feature in edge focus tasks, one particular attribute in node attributes matrix X for node focus tasks or a numerical interpretation of the state tuple for graph focus tasks.

2.2.2 Extrapolation and Interpolation Learning in Dynamic Graph

In real-world applications, we are facing the challenge of learning the network dynamic, namely the repetitive pattern in a dynamic graph's evolution over time. Dynamic graphs have one more dimension than static graphs have, which is time. The time dimension is usually stored as the timestamp when the observation of the graph or its components happens.

Let's denote a dynamic graph as $G_T = O_T$, where $T = [t_1 : t_n]$ is the time span from t_1 to t_n and is referred as the observation period. $O_T = \{o_{t_1}, o_{t_2}, \cdots, o_{t_n}\}$ is the set of observations which are performed within the observation period T. The observation could be a snapshot of graph $G_t = (V_t, E_t, \mathbf{X}_t)$ where V_t,E_t and $\mathbf{X}_t$ are the snapshot of the nodes, edges and node features at time t, a single node updating event $o_t = v_{i,t}$ or edge updating event $o_t = e_{\{i,j\},t}$ at time t. Supervised learning in dynamic graphs could be extrapolation learning, interpolation learning or time prediction.

In *Extrapolation Learning*, the purpose is to predict the label $Y_{t_{n+1}}$ at a future time based on the previous observations of a particular dynamic graph G_T and ground truth labels Y_T with the observation period $T = [t_1 : t_n]$. While in *Interpolation Learning*, the objective is to estimate the missing labels Y_{t_i} such that $t_i \in T$ and $Y_{t_i} \notin Y_T$. The extrapolation task predicts the future based on historical data, while the interpolation task gives an estimation for the past. Therefore, the interpolation task is mainly used in data imputation. As an example, in stock market analysis, if the purpose is to predict the future trend, then it is extrapolation learning; if the purpose is to fill some missing data, then it is interpolation learning.

In *Time Prediction*, the goal is to predict the time for a given incoming event. For example, predicting when the next crisis event would happen in *Integrated Crisis Early Warning System (ICEWS)* [86, 12]. The different learning tasks for supervised dynamic graph learning are summarized in Table 2.1

	Extrapolation	Interpolation	Time Prediction
Node Focus	Predict the target node's attribute in the future	Estimate the target node's missing attribute in the past	Predict when a given node updating event will happen
Edge Focus	Predict the target edge's status in the future	Estimate the target edge's status in the past	Predict when a given link will be added or deleted
Graph Focus	Predict the given dynamic graph's attribute in the future	Estimate the given graph's missing attribute in the past	Predict when the targeted graph will reach to a given state

Table 2.1: Supervised Learning task in Dynamic Graph

2.3 Taxonomy

2.3.1 Dynamic Graph Storage Model

A dynamic graph represents a graph evolving over time. Different kinds of graphs evolve differently. Some change very fast, but some others change very slowly. For example, a telephone SS7 voice network generates server logs in each node every second, while an interactive social network modelled from customer reviews has no updates for days. Zaki et el. [105] summarized that a dynamic graph could be modelled as either *Discrete Time Dynamic Graph (DTDG)* or *Continuous Time Dynamic Graph (CTDG)* based on how the temporal information is expressed regarding the evolution of the dynamic graph. DTDG is a list of snapshots, each of which keeps the graph status at a certain moment. Meanwhile, CTDG can be viewed as a stream of graph updating events. The definitions of these two graph representation models will be given out in Sec. 2.5 and 2.6 respectively. Such modelling paradigms are well adopted in the community [72, 100, 99, 43].

For a graph whose nodes or edges are frequently updated, it is more memory efficient to store it with DTDG modelling due to its snapshot-based representation [57]. However, it may induce some important temporal information loss if the observation frequency is not appropriately set. For example, if strong periodicity presents in nodes' updating events, each node's activity reaches its peak at noon every day. But the observation frequency is set to be every 24 hours, then such a periodicity pattern is never captured in the resulting snapshots. In comparison, CTDG can capture all temporal information as it is an event-based representation [57, 45].

Due to their pros and cons, the most important decision to make is which storage model to use. Such a decision must be made at the early stage of a dynamic graph learning use case. The selected storage model also limits the options of machine learning algorithms to only

those that are compatible. Algorithms designed specifically for DTDG, such as DySat [74] and STGCN [102], cannot be applied directly with CTDGs. Similarly, algorithms designed for CTDG, such as TGAT [99], cannot be applied with DTDGs either. In order to offer a fast reference for practitioners, the first hierarchy in our taxonomy to categorize different algorithms is the compatible graph storage model.

2.3.2 Encoder And Decoder Learning Framework

Graph learning algorithms are better considered to be under the encoder-decoder framework [32]. We follow the same framework when reviewing supervised dynamic graph learning algorithms. As shown in Fig. 2.3, the encoder turns the observations of a dynamic graph into its latent representation that is node-based. This latent representation is called graph embedding. The decoder decodes the generated graph embedding and gives the prediction or classification result. Under this framework, researchers can experiment with different encoder-decoder combinations to find the best one fitting the particular learning task [43].

The modification of the decoder to perform the three graph learning tasks is simple. The node focus task is straightforward since the embedding generated by the encoder is node-based. We can modify the decoder to take the concatenation of two nodes' embedding as input to perform the edge focus task, which is referred as the pair-wise decoder [32]. In the graph focus task, practitioners need to find an aggregation method to aggregate all nodes' embedding into the graph embedding. By modifying the decoder as described above, one can use the same encoder and decoder combination to perform all three dynamic graph learning problems. Therefore, algorithms fitting with an encoder-decoder framework are considered general-purpose learning methods.

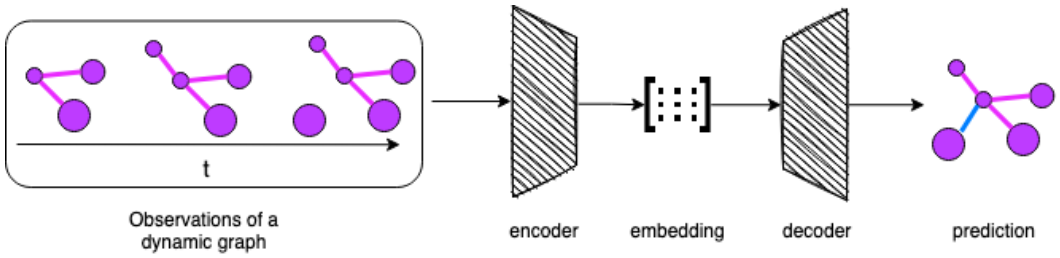


Figure 2.3: Encoder-Decoder Learning Framework

2.3.3 Implicit And Explicit Learning Model

The last hierarchy of the taxonomy in this survey paper is whether a given algorithm uses time as an input feature explicitly or implicitly. If time is used implicitly, it is not fed into the model as an independent feature. For example, the algorithm with a static graph encoder and LSTM decoder does not use time explicitly. The success of temporal learning relies on the time based ordering of the input snapshots and the regular observation frequency. We refer to this kind of learning model as *Implicit Time Learning Model*.

Meanwhile, if a given algorithm takes time as an independent input feature, it is referred to as *Explicit Time Learning Model*. Examples include TGAT [99] and TGNN[55] which use the time encoding Time2Vec [44] as a node feature. The explicit time learning model is capable of periodicity recognition and vector clock recognition, for which Implicit Time Learning Model is incapable.

Periodicity Recognition

In temporal data, periodicity is a frequently seen phenomenon. For example, 50% to 70% of human movements can be explained by periodic behaviour pattern [18]. Periodic patterns can be learnt by a sequential model in the DTDG setting. The learning model infers the underlying timestamps by the ordering and position of the input snapshots.

However, in CTDG, this is unfeasible without the explicit use of time. Kazemi et al. [44] proposed Time2Vec to encode time and help the downstream learning model recognize periodic patterns as well as linear time patterns.

TGAT [99] applies time2vec in message passing to assign more weight to neighbours with similar periodic patterns to target nodes. Explicit use of time with Time2Vec helps CTDG learning better learn periodical patterns.

Vector Clock Recognition

Information flowing between two given nodes needs to traverse their shortest path. The difference in path length and edge property results in the difference in the flowing information's arrival time at the target node. *Temporal Distance* between starting nodes and destination nodes is one metric to evaluate how much time is needed for the information to flow from the starting one to the destination [101]. Because the temporal distance from the target node to its neighbours has different values, the most up-to-date information with respect to its neighbours is sent at different timestamps. This phenomena is described by the concept *Vector Clock* [51]. Since different nodes have different vector clocks, they evolve differently from each other. *Temporal Point Process(TPP)* [71] is one of the temporal learning methods that are capable of learning the impact of vector clock. With TPP and the explicit use of time, a learning algorithm could recognize the impacts of a given node's vector clock on its evolution.

Moreover, the explicit use of time makes time prediction tasks feasible in dynamic graph learning. To the best of our knowledge, time prediction tasks could only be done with explicit time learning methods. Figure 2.1 shows the overall architecture of how this survey organizes different learning methods.

2.4 Temporal Pattern Learning

In this section, we present the Three Stages Recurrent Temporal Learning Framework . This framework describes a general form of dynamic graph learning algorithms and how they learn and apply the temporal pattern for different graph learning tasks. In some circumstances, node attributes are not available. Such dynamic graphs are called *non-attributed dynamic graph*. Conversely, dynamic graphs with attributes are called *attributed dynamic graph*. Previous works are limited to either attributed or non-attributed dynamic graphs [84, 23, 3]. To the best of our knowledge, Three Stages Recurrent Temporal Learning Framework is the only framework which could be generalized to both attributed and non-attributed dynamic graph learning. Subsection 2.4.1 presents the idea of the proposed framework and briefly introduces its three stages. Subsection 2.4.2, 2.4.3 and 2.4.4 explain the three stages in detail. Subsection 2.4.5 explains how to apply Three Stages Recurrent Temporal Learning Framework in attributed and non-attributed dynamic graphs. The equations presented in this section assume the given learning task is an extrapolation task. With some modification, they can be applied to interpolation tasks as well.

2.4.1 Three Stages Recurrent Temporal Learning Framework

Three Stages Recurrent Temporal Learning Framework describes how a learning algorithm learns the temporal pattern. A temporal pattern is a repetitive pattern in the given dynamic graph's evolution. A particular learning algorithm could learn the temporal pattern to perform those mentioned graph learning tasks. Given that the state of a dynamic graph G_t at time t is described by a timestamped state tuple $(E_t, \mathbf{X_t}, t)$, in which E_t and $\mathbf{X_t}$ is the edge connections and node attributes matrix observed at time t. The temporal pattern

could be expressed in the following function:

$$(\hat{E}_{t_{n+1}}, \hat{\mathbf{X}}_{t_{n+1}}, t_{n+1}) = \text{tp}(\{E_T\}, \{\mathbf{X_T}\}, T) \tag{2.1}$$

where $\{E_T\}$ and $\{\mathbf{X_T}\}$ are the set of observations of E and $\mathbf{X}$ in time period $T = [t_0, t_1, \cdots, t_n]$. $\hat{E}_{t_{n+1}}$ and $\hat{\mathbf{X}}_{\mathbf{t_{n+1}}}$ are the prediction of E and $\mathbf{X}$ for timestamp t_{n+1} based on the observed history.

To predict any missing entry in the state tuple, a learning algorithm needs an output function out$(\cdot)$ which takes the predicted state $(\hat{E}_{t_{n+1}}, \hat{\mathbf{X}}_{\mathbf{t_{n+1}}}, t_{n+1})$ and the known state $(E_{t_{n+1}} - e_{\{i,j\},t_{n+1}}), \mathbf{X}_{\mathbf{t_{n+1}}}, t_{n+1})$ as input. For example, in edge focus task, to predict the status of a given edge $e_{\{i,j\},t_{n+1}} \in E_{t_{n+1}}$ between v_i and v_j, the output function out$(\cdot)$ could be written as:

$$\begin{aligned} e_{\{i,j\},t_{n+1}} = \text{out}((\hat{E}_{t_{n+1}}, \hat{\mathbf{X}}_{t_{n+1}}, t_{n+1}), \\ (E_{t_{n+1}} - e_{\{i,j\},t_{n+1}}), \mathbf{X}_{\mathbf{t_{n+1}}}, t_{n+1})) \end{aligned} \tag{2.2}$$

A supervised dynamic graph learning algorithm needs to learn the temporal pattern tp$(\cdot)$ from the ground true history.

Three Stages Recurrent Temporal Learning Framework assumes tp$(\cdot)$ to be a three stages process such that it is a composite of the three functions:

$$\mathbf{X'}_{t_{n+1}} = \text{asu}(\{\mathbf{X}_T\}, T, t_{n+1}) \tag{2.3a}$$

$$\hat{\mathbf{E}}_{t_{n+1}} = \text{ap}(\{E_T\}, \{\mathbf{X}_T\}, \mathbf{X'}_{t_{n+1}}, T, t_{n+1}) \tag{2.3b}$$

$$\hat{\mathbf{X}}_{t_{n+1}} = \text{mp}(\hat{E}_{t_{n+1}}, \mathbf{X'}_{t_{n+1}}) \tag{2.3c}$$

$$\text{tp} = \text{mp} \circ \text{ap} \circ \text{asu} \tag{2.3d}$$

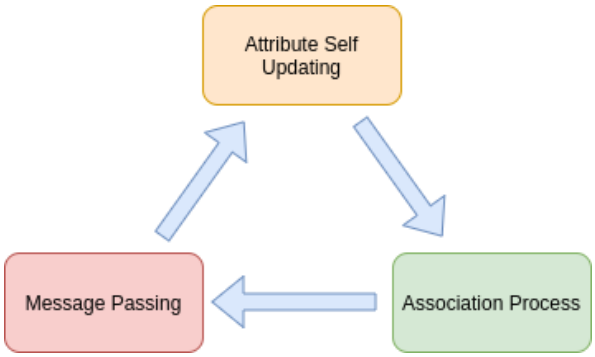


Figure 2.4: Three Stages Recurrent Temporal Learning Model

Functions asu(·), ap(·) and mp(·) each represents an intermediate stage in a dynamic graph's evolution which will be detailed in following subsections. The operation ∘ is function composition.

As shown in Fig. 2.4, the first stage is the *Attribute Self-Updating* defined as in Eqn. (2.3a). This stage captures the impact from the external factors to the graph evolution and gives out $\mathbf{X}'_{\mathbf{t_{n+1}}}$ as an estimation of X at t_{n+1}. Here we present a model with only node attributes self-updating. Readers can extend it to edge attributes and graph attributes self-updating with a similar schema. The change of attributes triggers the second stage, named *Association Process*. As in Eqn. (2.3b), the association process describes the evolution of the connection pattern. It generates new connection patterns based on the current connection pattern, timestamp and the self-updated attributes. The new connection pattern triggers the last stage, which is the *Message Passing*. As defined as Eqn. (2.3c), this stage integrates the impact of attributes self-updating and association process to generate the attributes for the next state. Consequently, the result of message passing would be the starting point of attributes self-updating in the next round of evolution. A dynamic graph's evolution can be described as a recurrent process of these three stages.

2.4.2 Attributes Self-Updating

The first stage in Three Stages Recurrent Temporal Learning Framework is the attributes self-updating. This stage captures the impact of external factors on the evolution of the given dynamic graph's attribute. Depending on the context, the attributes being impacted could be node attributes, edge attributes or graph attributes. The change of attributes alternatively drives the evolution of the given dynamic graph's connection pattern [84]. As an example, in a client-item knowledge graph, the unobserved change in the client's status would trigger the change in his interest.

***Definition* 4.** *Attributes Self Updating: the change of node, edge or graph's attributes resulted from external factors. Its equation asu*$(\cdot)$ *is defined in Eqn. (2.3a)*

As in our example, this stage only takes input of the history of the node attributes and the timestamps. Its output $\mathbf{X}'_{t_{n+1}}$ is the estimated node attributes for the next timestamp t_{n+1}.

2.4.3 Association Process

The change of attributes triggers the development of a new connection pattern. The process that describes the evolution of a dynamic graph's connection is called Association Process.

***Definition* 5.** *Association Process: The process that a particular dynamic graph develops, abandons or modifies the edges between its nodes. Its equation ap*$(\cdot)$ *is defined in Eqn. (2.3b).*

As shown in Eqn. (2.3b), the association process outputs the future connection pattern based on the given dynamic graph's evolution history, the estimated future attributes from the first stage, and the timestamps.

2.4.4 Message Passing

In graph analysis, we believe nodes are impacted by their neighbours. To learn how nodes are impacted by their neighbours, *Message Passing* is developed for Graph Neural Network(GNN)s [27, 31, 88].

***Definition* 6.** *Message Passing [27], also known as Affinity Propagation [91, 22] and Communication [85], is a local neighbourhood information aggregation method which updates node attributes by aggregating messages received from neighbouring nodes and the connected edges. Its equation $mp(\cdot)$ is defined in Eqn. (2.3c).*

Recent advancements in DTDG learning apply message passing to generate node-based graph embedding. The resulting graph embedding is considered to be a latent representation of the underlying graph's network structure and nodes/edges attributes [32, 97]. Therefore, it contains important information for graph-related machine learning tasks.

2.4.5 Generalization To Attributed And Non-attributed Dynamic Graphs

Three Stages Recurrent Temporal Learning Framework describes how attributed dynamic graph evolves by Eqn. (2.3). Supervised dynamic graph learning algorithms learn the temporal pattern F by optimizing the trainable weights in Eqn. (2.3) and (2.2) to best fit the input history.

When generalized to a non-attributed dynamic graph, the impact from node attributes is usually ignored since they are not available. The only driver considered in the graph's evolution is the association process. [57, 84]. In this case we can drop Eqn. (2.3a) from Eqn. (2.3d) and setting Eqn. (2.3b) to take input only $\{E_T\}$ and the timestamps as follows:

$$\hat{E}_{t_{n+1}} = \text{ap}(\{E_T\}, T, t_{n+1}). \tag{2.4}$$

TGN [72] and APAN [95] adopts Three Stages Recurrent Temporal Learning Framework by applying the node memory units to capture the attribute self-updating.

Know-Evolve [86], DyRep [85] and TGAT [99] adopt Three Stages Recurrent Temporal Learning Framework by setting Eqn. (2.3a) to always return the last observation in the input attributes history $\{\mathbf{X}_T\}$.

The learning of temporal pattern F is achieved by applying temporal learning algorithms in the dynamic graph encoder or directly in the decoder. Commonly-used temporal learning algorithms include RNN family neural networks, 1-D convolutional networks, time series analysis methods and attention networks. A recent trend is to explore the use of the TPP in temporal learning [86, 85].

2.5 Discrete Time Dynamic Graph Learning

The dynamic network represented by DTDG has a discretized time dimension. Each observation in the given dynamic graph G_T is expressed as a snapshot of the given graph attached with the observation timestamp. DTDG is defined as follows:

***Definition* 7.** *Discrete Time Dynamic Graph (DTDG): a dynamic graph $G_T = O_T$ for a time span $T = [t_1 : t_n]$is stored as DTDG, if each stored observation o_{t_i} in O_T is a snapshot of the given graph $o_{t_i} = (V_{t_i}, E_{t_i}, \mathbf{X}_{t_i})$ where V_{t_i}, E_{t_i} and $\mathbf{X}_{t_i}$ are nodes, edges and node features matrix observed at t_i .*

The data pipeline that makes the observation and stores the snapshot in a database is usually running at a regular frequency depending on the requirement, such as once per hour

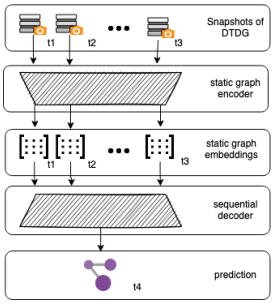
or once per day. The timestamps are usually as simple as ordered integers instead of actual date and time (e.g., $T = [1, 2, 3, \ldots, n]$).

The most seen form of supervised DTDG learning is the static graph encoder - sequential decoder framework. As shown in Fig. 2.5a, Algorithms following this framework use a static graph encoder to generate embedding for each snapshot and pass those embedding to a supervised sequential decoder for inference. Because the dynamic network has already been sliced into snapshots, there is no explicit usage of temporal information in the encoder. The encoder only captures the graph structure, property and attributes for each snapshot. Temporal information is learnt through the sequential decoder.

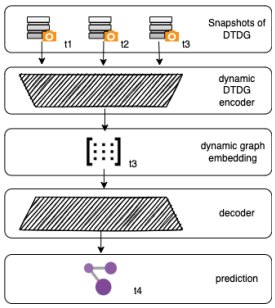
There are some emerging attempts to learn temporal patterns as well as graph topology and attributes in the encoder. These algorithms follow the dynamic graph encoder - simple decoder framework as shown in Fig. 2.5b. Instead of generating node-wise graph embedding for each snapshot in each inference, the dynamic graph encoder only generates one embedding recursively based on inputs in the past and the current input snapshot at each inference. Table 2.2 lists all encoders for supervised DTDG learning in this survey. So far as we summarize, all supervised DTDG learning methods do not learn the graph attribute self-updating process.

Time Model	Graph Type	Encoder
Implicit	non-attributed static graph	Shallow Embedding [8, 2, 14, 64, 94, 69, 103, 16, 29, 82]
		autoencoder Based Embedding [13, 90]
	attributed static graph	Message Passing GNN [76, 26, 107, 31]
		Graph Convolution Network [46, 47, 106, 31, 63, 56, 60]
		Graph Attention Network [88, 98, 74]
	dynamic graph	Kalman Filter Based Encoder [75]
		EvolveGCN [65]
		Spatial-Temporal Graph Convolution Network [102]
Explicit	N.A.	N.A.

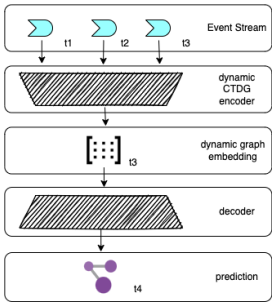
Table 2.2: DTDG Graph Encoders



(a) Static Graph encoder - Sequential Decoder Framework for DTDG



(b) Dynamic Graph Encoder - Simple Decoder Framework for DTDG



(c) Dynamic Graph Encoder - Simple Decoder Framework for CTDG

Figure 2.5: Different Encoder-Decoder Architectures for Dynamic Graph

2.5.1 Static Graph Encoder

Non-Attributed Static Graph Encoder

Static graph encoders could be categorized as non-attributed and attributed graph encoders. Non-attributed static graph encoders are not widely discussed in academia these days due to their incapability to leverage graph attributes. This motivates the recent exploration of attributed static graph embedding approaches. We will first review briefly the non-attributed

static graph embedding methods, and then provide an in-depth review of attributed static graph embedding methods. For those interested in more detail regarding non-attributed static graph representation learning algorithms, we refer them to read the recent work of Hamilton et al. [32], Cui et al. [19] and kazemi et al. [43].

A basic form of non-attributed static graph encoder is the shallow embedding approach, which aims to transform graph structure and property to node-level graph embedding [19]. However, graph attributes are not considered in shallow embedding. Shallow embedding includes:

- Matrix Factorization based approaches such as *Laplacian Eigenmaps* [8], *Graph Factorization* [2], *GraRep* [14], *HOPE* [64] and M-NMF [94].
- Random walk based approaches such as *DeepWalk* [69], *TADW* [103], *HARP* [16] and *node2vec* [29].
- Other approaches, such as *LINE* [82] .

A shallow embedding encoder is simply an embedding lookup based on node ids, and there is no parameter sharing between nodes. Hence, the computation is inefficient, and the trained encoder cannot be used for new graphs with unseen nodes [32].

To overcome these challenges and leverage graph attributes in the embedding generating process, multiple approaches are proposed to parameterized graph embedding (i.e., parameter sharing between nodes).

Deep Neural Graph Representation (DNGR) [13] and *Structural Deep Network Embeddings (SDNE)* [90] apply *autoencoder* [73] to map a high-dimension node similarity matrix to a low-dimension node embedding. These two approaches enable sharing of parameters, which can provide an efficient computation and be applied with unseen nodes.

Attributed Static Graph Encoder

To leverage graph attributes in static graph embedding generation, attributed static graph encoders are proposed. Attributed static graph encoders are based on *Graph Neural Network (GNN)* [76]. Such algorithms follow a message passing schema to aggregate neighbourhood information and generate embedding for the target node [26]. In this survey, we focus on some widely used GNN based attributed graph encoders, which are the foundation of dynamic graph encoders to review in later sections.

GNN assigns state $\mathbf{h}_{v_i}$ to node v_i of the input graph. Given mappings $N(v_i)$ to be all neighbors of node v_i, mapping $E(v_i)$ to be all edges connecting node v_i and its neighbors $N(v_i)$, $f_w(\cdot)$ to be the neighborhood information aggregating function, state $\mathbf{h}_{v_i}$ at the k layer is defined as in Eqn. (2.5).

$$\mathbf{h}_{v_i}^k = f_w(\mathbf{h}_{v_i}^{k-1}, E(v_i), \mathbf{h}_{N(v_i)}^{k-1}), \tag{2.5}$$

which can be written as Eqn. (2.6) when there is no positional information for the neighbours.

$$\mathbf{h}_{v_i}^k = \sum_{u \in N(v_i)} f_w(\mathbf{h}_{v_i}^{k-1}, e(v_i, u), \mathbf{h}_u^{k-1}). \tag{2.6}$$

The output state from the last layer n is the graph embedding $\mathbf{z}_{v_i}$:

$$\mathbf{z}_{v_i} = \mathbf{h}_{v_i}^n. \tag{2.7}$$

This state updating process is described as a message passing process in [26, 107]. In each iteration, the message from each node is passed through the edges to their neighbours. To learn local neighbourhood structure, k is required to be a small value, such as $k = 2$ in learning the 2-hop neighbourhood information. With Banach's fixed point theorem, the

state values will converge with the update of state in iterations [6, 26]. Therefore, to learn the global graph structure, we can continue the iterative updating until the change in the state value between two consecutive iterations is close to 0 as shown in Eqn. (2.8). When the state values converge, the global graph structure and property are embedded in the resulting embedding.

$$\mathbf{h}_{v_i}^{k} - \mathbf{h}_{v_i}^{k-1} \approx 0. \tag{2.8}$$

Stacking the converged state value z of each node together to produce Z, we obtain the node-wise embedding of the input graph. *GraphSAGE* [31] and *column network* [70] are following the same schema with different neighborhood information aggregation functions.

Inspired by GNN, *Graph Convolution Network (GCN)* [46] is a generalization of *Convolution Neural Network (CNN)* [48] to static graph data structure with spectral method. GCN applies graph Laplacian to generate the feature maps, which are shared within the whole graph as in CNN. Given the adjacency matrix $\mathbf{A}$, $\hat{\mathbf{A}} = \mathbf{A} + \mathbf{I}_N$, the diagonal degree matrix $\hat{\mathbf{D}} : \hat{\mathbf{D}}_{ii} = \sum_j \hat{\mathbf{A}}_{ij}$, the graph Laplacian is calculated as follows:

$$\mathbf{L} = \hat{\mathbf{D}}^{\frac{1}{2}} \hat{\mathbf{A}} \hat{\mathbf{D}}^{\frac{1}{2}}. \tag{2.9}$$

Moreover, a convolution layer with the output of d feature maps is calculated as follows:

$$\mathbf{Z} = \mathbf{LXW}, \tag{2.10}$$

where $\mathbf{X} \in \mathcal{R}^{|V| \times d}$ is the input node feature matrix with d features and $\mathbf{W} \in \mathcal{R}^{d \times d'}$ is the layer weight matrix to generate d' feature maps. The output $\mathbf{Z}$ of a hidden convolution layer is usually passed to an ReLU activation function, and the output of the activation function

will then be passed to the next layer as the input. As an example, a classic two layers GCN with softmax activation in the output layer has the form in Eqn. (2.11).

$$\mathbf{Z} = \text{softmax}(\mathbf{L} \cdot \text{ReLU}(\mathbf{LXW}^{\text{hidden}}) \cdot \mathbf{W}^{\text{out}}). \tag{2.11}$$

One example of GCN as a static graph encoder to learn the neighbourhood structure for each snapshot and a sequential decoder to learn the temporal pattern between snapshots is AddGraph [106].

Similar to GCN, *Graph Attention Network (GAT)* [88] applies the attention mechanism [4, 87] to determine the importance of neighboring nodes in the neighborhood aggregation for the target node:

$$\text{attn}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}(\text{score}(\mathbf{Q}, \mathbf{K}))\mathbf{V}, \tag{2.12}$$

where $\mathbf{Q}$ is the linear projection of the target node's input state $\mathbf{h}_{v_i}$ from the previous layer. $\mathbf{K}$ and $\mathbf{V}$ are the linear projections of the input state of the v_i's neighbouring nodes:

$$\begin{aligned} \mathbf{Q} &= \mathbf{h}_{v_i}\mathbf{W}_q \\ \mathbf{K} &= [\mathbf{h}_{v_1}, \mathbf{h}_{v_2}, \cdots, \mathbf{h}_{v_j}, \cdots, \mathbf{h}_{v_n}]\mathbf{W}_k, \forall v_j \in N(v_i) \\ \mathbf{V} &= [\mathbf{h}_{v_1}, \mathbf{h}_{v_2}, \cdots, \mathbf{h}_{v_j}, \cdots, \mathbf{h}_{v_n}]\mathbf{W}_v, \forall v_j \in N(v_i) \end{aligned} \tag{2.13}$$

The function score$(\cdot)$ calculates a score representing how well $\mathbf{Q}$ align to $\mathbf{K}$. The alignment scores for v_i's neighborhood $N(v_i)$ are then normalized by a softmax layer. The output of the whole attention layer is the product of the normalized alignment score and the linear projection $\mathbf{V}$.

In *Transformer* [87], the score function is the dot product of Q and K, while the score function is a leaky ReLU layer in *Graph Attention Network (GAT)* [88, 98]. GAT is used as

Storage Model	Time Model	Decoder
DTDG/CTDG	Implicit	Time Series Analysis Methods [78, 39, 30]
		RNN family [73, 35, 77, 63, 56, 60, 104, 81, 11]
		1-D Convolution Network [89]
		Positional Temporal Self Attention [74]
CTDG	Explicit	Temporal Point Process Decoder [86, 15]

Table 2.3: Decoders

a static graph encoder to learn DTDG in Dysat [74]. All attributed static graph encoders can be used to generate embedding in a supervised or unsupervised manner [31, 47, 46, 88, 70].

2.5.2 Sequential Decoder

In supervised DTDG learning, the temporal pattern is revealed by the change between snapshots along the time dimension. By sorting the static graph embedding generated from the series of snapshots according to their timestamps and treating them as sequential data, the temporal pattern can be learnt via a sequential decoder. However, the sequential decoder does not use time explicitly and is not capable of performing time predicting tasks. Table 2.3 lists different kinds of decoders summarized in this work.

Time Series Analysis Methods

DTDG can be considered as a time series of snapshots. Hence, traditional time series methods can be used naturally as decoders to learn temporal patterns. *Exponential Moving Average (EMA)* and *Auto-Regressive Integrated Moving Average (ARIMA)* are two widely-used methods, which are used as decoders in previous works [78, 39, 30]. As an example, Eqn. (2.14) shows the equation of EMA with a predefined smoothing factor $\alpha \in (0, 1)$.

$$\mathbf{Z}_{t_{n+1}} = \sum_{i=0}^{n} \alpha(1-\alpha)^i \mathbf{Z}_{t_{n-i}}. \tag{2.14}$$

Recurrent Neural Network (RNN) family

RNN based methods are frequently-used as supervised sequential decoders and are capable of learning temporal correlation. A basic form of RNN decoders is defined by a recurrent state function which takes the previous state and the current embedding as input [73]:

$$\mathbf{h}_{v_i,t_n} = \text{RNN}(\mathbf{h}_{v_i,t_{n-1}}, \mathbf{z}_{v_i,t_n}). \tag{2.15}$$

The generated state is used as the input in the output unit, which is usually a *Feed forward Neural Network (FNN)*:

$$o_{v_i,t_n} = \text{FNN}(\mathbf{h}_{v_i,t_n}). \tag{2.16}$$

One frequently used RNN model is the *Long Short Term Memory (LSTM)* [35], which better learns the long-term temporal pattern. For example, Seo et al. [77] applied the spectral GCN [21] as static graph encoder and LSTM as decoder; Narayan et al. [61] used a different GCN [63] as static graph encoder and LSTM as decoder; Manessi et al. [56] and Mohanty et al. [60] also used different version of GCN as static graph encoder and LSTM as decoder to perform dynamic graph learning. Yuan et al. [104] apply message passing GNN as a static graph encoder and a four gates LSTM as a sequential decoder to learn the dynamic graph constructed from video frames and performs object detection. DyGGNN [81] uses a gated graph neural network [54] as encoder and LSTM as decoder. Bogaets et al. [11] apply CNN as a static graph encoder and LSTM as a sequential decoder in traffic forecasting for the road network in the city of Xi'an.

1-D Convolution Neural Network (Conv1d)

Conv1d is used in time series analysis [52]. Therefore, it is naturally used as a sequential decoder to learn the temporal pattern. Unlike RNN families, Conv1d only learns short-term temporal patterns in the given time frame. The periodicity of those short-term temporal patterns could be learnt by stacking multiple Conv1d layers. By carefully setting the size of each feature map, Conv1d inputs the embedding of a given node for n most recent snapshots $[\mathbf{z}_{v_i,t_1}, \mathbf{z}_{v_i,t_2}, \cdots, \mathbf{z}_{v_i,t_n}]$, and generates the decoded state $\mathbf{h}_{v_i,t_n}$ for node v_i at time t_n:

$$\mathbf{h}_{v_i,t_n} = \text{Conv1d}([\mathbf{z}_{v_i,t_1}, \mathbf{z}_{v_i,t_2}, \cdots, \mathbf{z}_{v_i,t_n}]). \tag{2.17}$$

The decoded state is then passed to the output unit as in Eqn. (2.16).

GraphTCN [89] applies a GAT based static graph encoder and Conv1d as the sequential decoder to learn the spatial and temporal information in Human Trajectory Prediction.

Temporal Self-Attention (TSA)

The attention mechanism is proved to perform very well in sequential data learning [4, 17]. As in Eqn. (2.18), given the node embedding $\mathbf{Z}_{v_i}$ of the target node v_i, each element encodes a snapshot from the observation period $[t_1, t_2, \cdots, t_{n-1}, t_n]$, TSA uses each element in $\mathbf{Z}_{v_i}$ as a query to attend over the whole input history $\mathbf{Z}_{v_i,[t_1:t_n]}$ to generate the temporal graph embedding for the corresponding snapshot. The output of a TSA layer $\mathbf{H}_{v_i}$ has the same time dimension as the input, such that it is feasible to stack multiple TSA layers to learn

the evolution of its local neighbourhood structure and attributes over time.

$$\begin{aligned}
\mathbf{Z}_{v_i} &= [\mathbf{z}_{v_i,t_1}, \mathbf{z}_{v_i,t_2}, \cdots, \mathbf{z}_{v_i,t_n}] \\
\mathbf{Q}_{t_i} &= (\mathbf{z}_{v_i,t_i} + \mathbf{P}_{t_i})\mathbf{W}_q, \\
\mathbf{K} &= (\mathbf{Z}_{v_i} + \mathbf{P})\mathbf{W}_k, \\
\mathbf{V} &= (\mathbf{Z}_{v_i} + \mathbf{P})\mathbf{W}_v, \\
\mathbf{h}_{v_i,t_i} &= \text{attn}(\mathbf{Q}_{t_i}, \mathbf{K}, \mathbf{V}), \\
\mathbf{H}_{v_i} &= [\mathbf{h}_{v_i,t_1}, \mathbf{h}_{v_i,t_2}, \cdots, \mathbf{h}_{v_i,t_n}].
\end{aligned} \tag{2.18}$$

As in Eqn. (2.12), Dysat [74] applied TSA as the sequential decoder in DTDG learning. Their score function is shown in Eqn. (2.19).

$$\text{score}(\mathbf{Q}_{t_i}, \mathbf{K}) = \frac{\mathbf{Q}_{t_i}\mathbf{K}^{\mathbf{T}}}{\sqrt{d'}}. \tag{2.19}$$

To the best of our knowledge, there is no explicit time sequential decoder proposed for supervised DTDG learning. Because the snapshots are taken at a regular frequency, temporal information is revealed in the ordering and the position of the snapshots. However, without using time as a learning feature, the implicit time sequential decoder is not capable of performing time prediction tasks as discussed in Sec. 2.2.

2.5.3 Dynamic DTDG Encoder

Implicit Time DTDG Encoder

Recent developments commonly use attributed dynamic graph embedding as an encoder to learn both graph structures and temporal correlations. The generated dynamic graph

embedding is then fed to a simple supervised predictive model as a decoder to conduct prediction.

Kalman Filter Based Encoder *Kalman Filter* [42], also called *Linear Quadratic Estimation*, is widely used in sensor signal refinement. It is efficient in handling uncertainty caused by random external factors. When considering node properties as sensor measures, Kalman Filter can be used to generate dynamic node embedding. A hidden state matrix $\mathbf{H}_{t-1}$ at time $t-1$ is formed by stacking the hidden state of each node in the graph or the local neighbourhoods. Kalman Filter based encoder is a two-step recurrent process that includes the prediction step and the hidden state updating step. Given the hidden state $\mathbf{H}_{t-1}$ at snapshot $t-1$ and its estimated covariance matrix $\hat{\mathbf{P}}_{t-1}$, the embedding matrix $\mathbf{Z}_t$ at time t and its predicted covariance matrix $\mathbf{P}_t$ is calculated as in Eqn. (2.20), where $\mathbf{W}$ and $\mathbf{B}$ are trainable parameters, and $\mathbf{Q}_t$ is a random noise drawn from a zero-mean Gaussian distribution, and N_{t-1} is a control factor that could be simply 0 as in Sarkar et al. [75] or neighborhood embedding aggregation.

$$\begin{aligned} \mathbf{Z}_t &= \mathbf{W}\mathbf{H}_{t-1} + \mathbf{B}\mathbf{N}_{t-1}, \\ \mathbf{P}_t &= \mathbf{W}\hat{\mathbf{P}}_{t-1}\mathbf{W}^{\mathbf{T}} + \mathbf{Q}_t. \end{aligned} \tag{2.20}$$

Once a new observation of node attributes $\mathbf{X}_t$ at time t is obtained, the Kalman gain $\mathbf{K}_t$ is defined as in Eqn. (2.21), where $\mathbf{J}$ is a trainable parameter.

$$\mathbf{K}_t = \mathbf{P}_t\mathbf{J}^{\mathbf{T}}(\mathbf{J}\mathbf{P}_t\mathbf{J}^{\mathbf{T}} + \mathrm{Cov}(\mathbf{X}_t))^{-1}. \tag{2.21}$$

The hidden state $\mathbf{H}_t$ and the estimated covariance $\hat{\mathbf{P}}_t$ are updated as follows:

$$\begin{aligned} \mathbf{H}_t &= \mathbf{Z}_t + \mathbf{K}_t(\mathbf{X}_t - \mathbf{J}\mathbf{Z}_t), \\ \hat{\mathbf{P}}_t &= \mathbf{P}_t - \mathbf{K}_t\mathbf{J}\mathbf{P}_t. \end{aligned} \tag{2.22}$$

EvolveGCN The idea of EvolveGCN [65] is simple and interesting. In order to to make the model adaptable to newly added nodes, EvolveGCN focuses on training an RNN model to learn the temporal dynamic presented in the underlying GCN. Namely, the parameters in the underlying GCN are not learned. They are predicted by an RNN model. There are two versions of the EvolveGCN unit, which are the hidden unit *EGCU-H* and the output unit *EGCU-O*. *EGCU-H* takes the input of last layers output states $\mathbf{H}_t^{l-1}$ and the parameters from the last time step $\mathbf{W}_{t-1}^l$, and then outputs the parameter for the current time step $\mathbf{W}_t^l$ as shown in Eqn. (2.23).

$$\begin{aligned} \mathbf{W}_t^l &= \mathrm{GRU}(\mathbf{H}_t^{l-1}, \mathbf{W}_{t-1}^l), \\ \mathbf{H}_t^{l+1} &= \mathrm{GNN}(\mathbf{A}_t, \mathbf{H}_t^l, \mathbf{W}_t^l). \end{aligned} \tag{2.23}$$

Similarly, *EGCU-O* calculates the parameter for the underlying GCN at the current time step by LSTM but takes only the parameter from the last time step as input as follows:

$$\begin{aligned} \mathbf{W}_t^l &= \mathrm{LSTM}(\mathbf{W}_{t-1}^l), \\ \mathbf{H}_t^{l+1} &= \mathrm{GNN}(\mathbf{A}_t, \mathbf{H}_t^l, \mathbf{W}_t^l). \end{aligned} \tag{2.24}$$

Spatial Temporal Graph Convolutional Network (ST-GCN) ST-GCN is developed to learn both the spatial and temporal patterns for human action recognition [102]. An ST-GCN layer is composed of two operations: spatial convolution and temporal convolution.

Spatial convolution learns the graph structure pattern. It adds a partitioning strategy to spectral GCN as described in Eqn. 2.10 to assign different weights to the nodes in different partitions so as to learn the feature importance for different partitions based on their spatial information. There are three partitioning strategies proposed in [102]:

- *uni-labelling*: all nodes are assigned in the same partition.
- *distance partitioning*: the target node v_i is assigned to partition 0, and the partitioning of its neighbours is determined by the length of their shortest paths to v_i.
- *spatial configuration*: the partition is determined by the given node's distance to the graph's centroid, as shown in Eqn. (2.25), where r_i is the distance between v_i and the graph centroid, and $l_{ti}(\cdot)$ is a function whose output is the partition label of the input node v_j at time t in the state calculation for node v_i.

$$l_{ti}(v_{tj}) = \begin{cases} 0 & \text{if } r_j = r_i \\ 1 & \text{if } r_j < r_i \\ 2 & \text{if } r_j > r_i \end{cases} \tag{2.25}$$

After the partitions are generated, the graph Laplacian will be broken down accordingly. With uni-labelling partitioning, the resulting graph Laplacian is $I_N + A$ which is exactly the same as the GCN proposed by Kipf et al. [47]. With distance partitioning and spatial partitioning, the graph Laplacian is dismantled into multiple tensors A_j according to the partition label such that the sum of those tensors equals $I_N + A$ as follows:

$$\mathbf{I}_N + \mathbf{A} = \sum_j \mathbf{A}_j. \tag{2.26}$$

Each A_j has its own learnable weight M, and the GCN encoder in ST-GCN is calculated as shown in Eqn. (2.27) where $\otimes$ denotes the element-wise multiplication.

$$\mathbf{Z} = \sum_j \mathbf{D}_j^{\frac{1}{2}}(\mathbf{A}_j \otimes \mathbf{M}_j)\mathbf{D}_j^{\frac{1}{2}}\mathbf{X}\mathbf{W}. \tag{2.27}$$

The resulting embeddings for the input snapshots are ordered based on their timestamps as the output embedding.

As shown in Fig. 2.6, The temporal convolution performs 2-D convolution for each node along its time dimension T and feature dimension D to learn the temporal pattern.

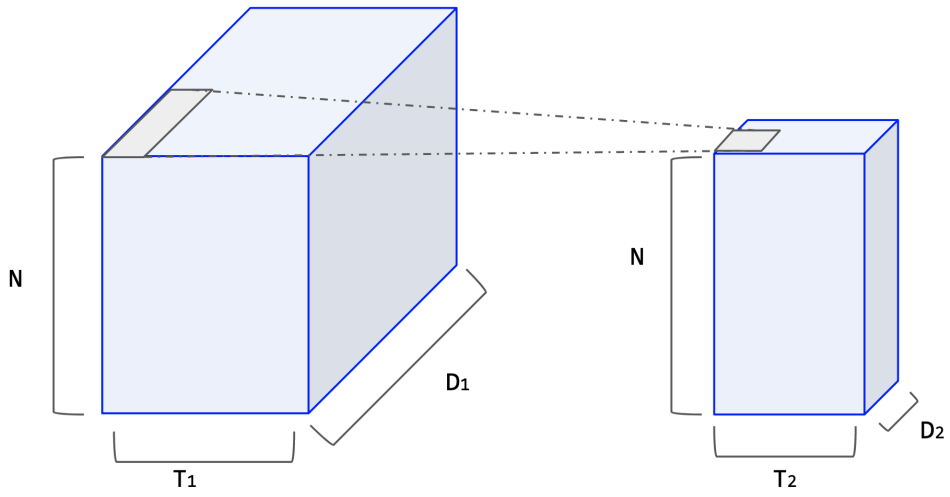


Figure 2.6: Temporal Convolution

An ST-GCN layer can have multiple temporal and spatial convolutions. Multiple ST-GCN layers can be stacked together to construct a deep dynamic graph encoder for better expressive power. The implementation in Yan et al. [102] ensembles an ST-GCN layer with one spatial convolution in between two temporal convolutions.

Explicit use of time in DTDG has not been explored in the community to the best of our knowledge. Neither does the time prediction task for DTDG. It is an interesting direction for future work to develop an explicit time learning model for DTDG and perform time prediction tasks.

2.6 Continuous Time Dynamic Graph Model

DTDG is a well-explored dynamic graph model which offers plenty of learning algorithms for downstream applications. However, as discussed above, it is possible to lose important temporal information when DTDG storage models are applied with inappropriate observation frequency. To preserve all temporal information, we can use *Continuous Time Dynamic Graph(CTDG)* storage model. CTDG is also called a streaming graph. Under this storage model, the dynamic graph is modelled and stored as the graph updating event stream. Because all the changes and their timestamps are kept in the database, there is no loss of temporal information.

***Definition* 8.** *Continuous Time Dynamic Graph (CTDG): a dynamic graph G_T with $T = [t_0 : t_n]$ is regarded as a CTDG model if it is stored as $G_T = (G_{t_0}, O_{[t_1:t_n]})$ with $O_{[t_1:t_n]}$ to be a collection of timestamped graph updating events observed during the time span $[t_1 : t_n]$, G_{t_0} to be its initial state at t_0. Each event $o_{t_i} \in O_{[t_1:t_n]}$ could be either a node updating event x_{v_i,t_i} or edge updating event $e_{\{i,j\},t_j}$.*

CTDG learning algorithm aims to learn the network evolution embedded in the events stream. As shown in Fig. 2.5c, the framework for CTDG learning algorithms is very similar to that of the dynamic DTDG encoder (the simple decoder framework in Fig. 2.5b). In this framework, the input is the observed updating event stream, and then the dynamic CTDG encoder transforms the input event stream to a node-wise graph embedding that learns the

graph's temporal pattern in its evolution. The encoder learns the regular temporal pattern as described by Eqn. (2.1). Hence it can not only be used directly with common decoders such as MLP [25] and SVM [80] but also be used with sequential decoders introduced in Sec. 2.5. Moreover, this framework is commonly applied to analyze interaction networks [86, 15], transaction networks [95], and knowledge graphs in recommendation systems [95].

There are three major challenges in the supervised learning of CTDG: *Event Expiration*, *Computational Exhaustive in Adjacency Matrix Retrieval*, and *Temporal Information Learning*.

Event Expiration

This is the staleness problem proposed by Kazemi et al. [43] for CTDG learning. How can we determine if long-ago updating events have large impacts on the current nodes? For example, the relations between users in a social network are defined by their phone call activities, and a phone number is previously abandoned and is recently assigned to a new user. In this case, the previous events for the node represented by this number should be expired and no longer reflect its current owner's social relationship, and this node should be counted as a new user without history.

Computational Exhaustive in Adjacency Matrix Retrieval

CTDG is stored as a collection of updating events. To obtain its adjacency matrix, we need to scan the whole history and construct the relation between nodes to fill in the cells in the adjacency matrix for each source and destination node pair according to the observed events and their timestamp. However, the computing resources are costly, so we try to avoid the adjacency matrix construction in CTDG learning.

Temporal information Learning

Each observation in CTDG has its own timestamp, and hence rich temporal information can be learnt by analyzing their timestamps. The challenge of learning the temporal information is how we can properly use the timestamps as features to learn from.

In the following sections, we will review different dynamic graph encoders for CTDG and discuss how these graph encoders tackle the aforementioned challenges. All summarized CTDG encoders are listed in Table 2.4:

Time Model	Graph type	Encoder
Implicit	Non-attributed Dynamic Graph	Temporal Random Walk-Based Encoder [62, 20, 7, 85]
Explicit	Attributed Dynamic Graph	Temporal Attention Based Encoder [99, 100]
		RNN based Encoder [72, 49, 50, 86, 85, 95]

Table 2.4: CTDG Graph Encoders

2.6.1 Implicit Time CTDG Encoder

The temporal process regarding the graph topology can be learnt through the *Temporal Random Walk-Based Encoder*. Temporal Random Walk is defined as a random walk performed respecting the timestamp of the edge updating events [57, 36, 45, 10]. Once the sets of temporal walks are sampled, random walk-based static graph encoders can be used to generate the embedding. Nguyen et al. [62] propose a temporal random walk-based encoder for non-attributed dynamic graphs with three strategies to select the next hop in a walk. De Winter et al. [20] and Bastas et al. [7] convert the CTDG to DTDG and perform temporal random walks. Since random walk-based approaches are generally applied in non-attributed dynamic graphs, practitioners have to combine them with a decoder that utilizes the graph attributes for attributed dynamic networks. One potential direction of future works can extend temporal random walk-based encoders for attributed dynamic graphs by biasing the hop selection based on TPP. The temporal attentive aggregation for neighbourhood message passing in DyREP [85] is an example in which the maximum walk length is 1, and the probability of the next hop is calculated based on TPP.

2.6.2 Explicit Time CTDG Encoder

CTDG is stored as a stream of observed updating events $O(T)$, which could be viewed as sequential data. Therefore sequential learning models are naturally used in dynamic CTDG encoder to transform a given node's updating events to its embedding. One such kind of encoder is the temporal attention based encoder.

Temporal Attention Based CTDG Encoder

Inspired by the success of the network attention mechanism in learning sequence data [87], Xu et al. proposed the *Temporal Attention* mechanism as the dynamic graph encoder in Temporal Graph Attention Network (TGAT). It assumes that more critical temporal information is revealed in the relative time span, compared to the absolute time value. However, attributes self-updating is not considered in its temporal learning framework.

With this assumption, Temporal Attention applies Time2Vec [44] to capture critical temporal information from the dynamic graph. Time2Vec aims to generate a simple vector representation of time so as to enable different learning algorithms to learn the temporal correlation as well as the periodicity with the explicit use of time. Given a scalar notion of time Δt, which could be the time difference in the CTDG setting, its naive Time2Vec encoding $\text{t2v}(\Delta t)$, is a vector of predefined size d and calculated as:

$$\text{t2v}(\Delta t)[i] = \begin{cases} w_i \Delta t + \varphi_i & \text{if } i = 1, \\ f(w_i \Delta t + \varphi_i) & \text{if } 1 < i \leq d, \end{cases} \tag{2.28}$$

where w_i and φ_i are trainable weight or predefined weight, and f is a periodic function, such as $\sin(\cdot)$ and $\cos(\cdot)$. Time2Vec helps the learning model to learn temporal correlation by simple linear time projection when $i = 1$, and it helps learn the periodicity of time by kernel learning when $1 < i \leq d$ [99].

TGAT applied a modified version of Time2Vec to calculate the functional encoding of a given node v_i at time t:

$$\mathbf{h}_{v_i,v_j} = \mathbf{x}_{v_j,t_j}||\mathbf{e}_{ij,t_j}||\text{t2v}(t_i - t_j) \tag{2.29a}$$

$$\begin{aligned}\mathbf{H}_{v_i} &= [\mathbf{h}_{v_i,v_i}, \mathbf{h}_{v_i,v_1}, \cdots, \mathbf{h}_{v_i,v_j}, \cdots, \mathbf{h}_{v_i,v_n}]^{\mathbf{T}} \\ &\forall v_j \in N(v_i)\end{aligned} \tag{2.29b}$$

$$\begin{aligned}\text{t2v}(\Delta t) = \sqrt{\frac{1}{d}}[&\cos(w_1\Delta t), \sin(w_1\Delta t), \cdots, \\ &\cos(w_d\Delta t), \sin(w_d\Delta t)]\end{aligned} \tag{2.29c}$$

where $\mathbf{x}_{v_i,t_i}$ is the node feature vector for node v_i at time t_i; $\mathbf{e}_{ij,t_j}$ is the edge feature vector for the edge between node v_i and v_j at time t_j. In Eqn. (2.29a), if $v_j = v_i$, $\mathbf{e}_{ij,t_j}$ is vector of zeros.

With the functional encoding, temporal attention calculates the query, key and value for v_i at time t_i as:

$$\begin{aligned}\mathbf{Q}_{v_i,t_i} &= [\mathbf{H}_{v_i}]_0\mathbf{W}_Q, \\ \mathbf{K}_{v_i,t_i} &= [\mathbf{H}_{v_i}]_{1:n}\mathbf{W}_K, \\ \mathbf{V}_{v_i,t_i} &= [\mathbf{H}_{v_i}]_{1:n}\mathbf{W}_V,\end{aligned} \tag{2.30}$$

and the output state of the target node v_i is given by Eqn. (2.12):

$$\mathbf{z}_{v_i,t_i} = \text{attn}(\mathbf{Q}_{v_i,t_i}, \mathbf{K}_{v_i,t_i}, \mathbf{V}_{v_i,t_i}). \tag{2.31}$$

RNN based CTDG Encoder

As discussed in Sec. 2.4, external factors have a major contribution to a dynamic graph's evolution. *RNN-based CTDG encoder* follows the Three Stages Recurrent Temporal Learning Framework to capture the external factors contributing to the node attribute self-update. RNN-based encoder first generates an impression from the observed events related to the target node by a memory function and then maintains its memory related to the target node by feeding the impression to a sequential model. Node embedding is generated from the maintained memory optionally, together with other factors, such as node embedding from other kinds of encoders, embedding from the observed events, and embedding from the timestamps.

Temporal Graph Neural Network (TGN) [72] provides a general framework for RNN based encoder. *TGN* consists of two components which are the memory component and the embedding component. The memory component represents the model's memory for a given node's history. we can denote the memory of node v_i at time t by vector $\mathbf{s}_{v_i,t}$. $\mathbf{s}_{v_i,t}$ is updated when an updating event involving node v_i is encountered. If the event is a node-wise event with new node attribute vector $\mathbf{x}_{v_i,t}$, the message to $\mathbf{s}_{v_i,t}$ will be calculated as shown in Eqn. (2.32), where t^- is the timestamp for the last observation for node v_i before t and Δt is the time span between t^- and t:

$$\mathbf{m}_{v_i,t} = \mathrm{msg}_n(\mathbf{s}_{v_i,t^-}, \Delta t, \mathbf{x}_{v_i,t}) \tag{2.32}$$

For an interaction event with new edge feature vector $\mathbf{e}_{ij,t}$ indicating node v_i as source and v_j as destination, the message is calculated as shown in Eqn. (2.33a) and (2.33b), respectively.

$$\mathbf{m}_{v_i,t} = \text{msg}_s(\mathbf{s}_{v_i,t^-}, \mathbf{s}_{v_j,t^-}, \Delta t, \mathbf{e}_{ij,t}, \tag{2.33a}$$

$$\mathbf{m}_{v_j,t} = \text{msg}_d(\mathbf{s}_{v_j,t^-}, \mathbf{s}_{v_i,t^-}, \Delta t, \mathbf{e}_{ij,t}). \tag{2.33b}$$

For an undirected network, msg_s and msg_d share the same parameters.

Once the messages for all input events with the target node v_i involved are generated, they are aggregated as described in Eqn. (2.34), where $t^- < t_1, \ldots, t_b <= t$:

$$\mathbf{s}_{v_i,t} = \text{mem}(\mathbf{s}_{v_i,t^-}, \text{agg}(\mathbf{m}_{v_i,t_1}, \ldots, \mathbf{m}_{v_i,t_b})) \tag{2.34}$$

The $\text{agg}(\cdot)$ refers to an aggregation function, which can be a trainable deep learning layer (e.g., RNN, LSTM) or a simple aggregation without trainable weight (e.g., the mean of those messages, the most recent message). The $\text{mem}(\cdot)$ function is a deep learning layer representing the memory of the model and should be selected from the RNN family. The output of the memory component $s_{i,t}$ can be used directly as the resulted node embedding as in Jodie [49, 50], Know-Evolve [86] and DyREP [85].

However, as proposed by Kazemi et al. [43], there is a so-called memory staleness problem for nodes that are not active for a relatively long time, depending on the context. The memory $\mathbf{s}_{v_i,t^-}$ of this kind of node is not updated for a long time. Once a new event arrives, the outdated memory has the same impact as a recent memory. When calculating the new memory $\mathbf{s}_{v_i,t}$, this is not desirable when the events that happened a long time ago have not much impact on the graph's evolution. For example, two users with past frequent connections may not be currently connected in a call social network since one of the users changed the phone number. A new event for an abandoned number represents that this number is recycled and used by a different user, and the history for this number is no longer

meaningful. To overcome this challenge, TGN proposed the embedding component, which uses the time embedding [44, 99] as one of the input features that could be trained to recognize the impact of staled memory.

The embedding component $\mathbf{z}_{v_i,t}$ has a general form as the following message passing schema:

$$\mathbf{z}_{v_i,t} = \sum_{v_j \in N_t(v_i)} f(\mathbf{s}_{v_i,t}, \mathbf{s}_{v_j,t_j}, e_{ij,t_j}, \mathbf{x}_{v_i,t}, \mathbf{x}_{v_j,t}), \tag{2.35}$$

where $f(\cdot)$ is a neighbour node aggregation function and t_j is the timestamp that the last edge updating event observed between v_i and v_j.

The embedding component $\mathbf{z}_{v_i,t}$ has different implementations and can generate embedding without neighbourhood information aggregation. For example, it could be as simple as just the identify function of the memory $\mathbf{s}_{v_i,t}$; the time projection used in Jodie [50]:

$$\mathbf{z}_{v_i,t} = (1 + \Delta t w) \cdot \mathbf{s}_{v_i,t}. \tag{2.36}$$

A MLP-TGAT network structure takes the states of the target node v_i and its neighbors from the last layer and the Time2Vec embedding of their last updating events' time stamps $\text{t2v}(t - \mathbf{T}_{N(v_i)})$ as input, the input state $\mathbf{h}^{(1)}_{v_i,t}$ of the first layer is the memory $\mathbf{s}_{v_i,t}$ and the output state $\mathbf{h}^{(l)}_{v_i,t}$ of the last layer would be the resulting embedding:

$$\begin{aligned}
\mathbf{h}^{(1)}_{v_i,t} &= \mathbf{s}_{v_i,t} \\
\hat{\mathbf{h}}^{(l)}_{v_i,t} &= \text{TGAT}(\mathbf{h}^{(l-1)}_{v_i,t}, \text{t2v}(0), \mathbf{H}^{(l-1)}_{N(v_i)}, \text{t2v}(t - \mathbf{T}_{N(v_i)})) \\
\mathbf{h}^{(l)}_{v_i,t} &= \text{MLP}(\mathbf{h}^{(l-1)}_{v_i,t}, \hat{\mathbf{h}}^{(l)}_{v_i,t})
\end{aligned} \tag{2.37}$$

Similarly, Temporal Graph Sum is proposed in TGN [72]:

$$\begin{aligned}\hat{\mathbf{h}}_{v_i,t}^{(l)} &= \text{ReLU}(\sum_{v_j \in N(v_i)} \mathbf{W}_1^{(l)}(\mathbf{h}_{v_j,t_j}^{(l-1)}, \text{t2v}(t-t_j))),\\ \mathbf{h}_{v_i}^{(l)} &= \mathbf{W}_2^{(l)}(\mathbf{h}_{v_i,t}^{(l-1)} || \hat{\mathbf{h}}_{v_i,t}^{(l)}).\end{aligned} \tag{2.38}$$

APAN [95] applies an asynchronous mail propagator to overcome the out-of-order event arrival issue in TGN's node memory units design. In online training, graph updating events are not guaranteed to arrive in timestamp order. This will bring instability to RNN based CTDG encoders such as TGN. The asynchronous mail propagator fixes this issue by storing the incoming events in their timestamp order.

2.6.3 Explicit Time CTDG Decoder

As discussed, the decoder in CTDG learning can be a simple supervised classifier, such as MLP [25] and SVM [80]. The application of sequential decoders is introduced in Sec. 2.5. Temporal Point Process Based Decoder in CTDG learning is introduced in this section.

Temporal Point Process Based Decoder Know-Evolve [86] and TDIG-MPNN [15] apply TPP as a decoder in edge focus tasks and time predicting tasks with CTDG setting.

Instead of modeling the set of observations $O_{[0:t]}$ as sequential data, temporal point process models $O_{[0:t]}$ as a random process with parameters the input time t and the observations $O_{[0:t^-]}$ before t [71]:

$$f(t, O_{[0:t^-]}) = \lambda(t, O_{[0:t^-]}) \text{exp}(-\int_{t^-}^{t} \lambda(t, O_{[0:t^-]}) dt), \tag{2.39}$$

where $f(t, O_{[0:t^-]})$ is the probability density function representing an event occurs at time t given the previous observations. t^- is the time for the last observation right before t. $\lambda(\cdot)$

is the conditional intensity function. Its form depends on the temporal pattern to capture, such as the Poisson process, Hawkes process [34], Self-correcting process [40], Power Law and Rayleigh process [59]. In Know-Evolve, the conditional intensity function $\lambda(\cdot)$ for the target edge $e_{i,j}$ at time t is described as follows:

$$\lambda_{i,j}(t|t^-) = \exp(\mathbf{z}^{\mathbf{T}}_{v_i,t^-} \cdot \mathbf{W}_{i,j} \cdot \mathbf{z}_{v_j,t^-})\dot{(}t - t^-), \tag{2.40}$$

where $\mathbf{W}_{i,j} \in \mathcal{R}^{d\times d}$ is the unique trainable parameter regarding the edge $e_{i,j}$. For a Rayleigh process, the survival term in Eqn. (2.39) is calculated as:

$$\exp(-\int_{t^-}^{t} \lambda_{i,j}(t|t^-)dt) = \lambda_{i,j}(t|t^-) \cdot (t^2 - (t^-)^2). \tag{2.41}$$

In the inference, practitioners can estimate the probability that an event happens for the given edge in the next moment by the product of the resulting probability density and the time duration in interest to perform edge focus task and time predicting task. Graph Hawkes Neural Network applies the Hawkes process in its decoder [33] in a similar manner. It would be interesting for future work to extend the TPP-based decoder for node focus tasks as well as graph focus tasks. Also, applying a TPP-based decoder in DTDG learning for time predicting tasks is an interesting future work as well.

2.7 Challenges And Future Works

In this section, we will highlight some challenges and interesting future works in supervised learning for both DTDG and CTDG.

Explicit Time DTDG Learning

DTDG is a very well-explored dynamic graph model. Because its snapshot-based representation fits naturally with static graph encoder, most works focus on performing node focusing, edge focusing and graph focusing tasks with the framework of static graph encoder and sequential decoder. However, since time is implicitly learnt in the decoder with this framework, it is not capable of performing time predicting tasks. To the best of our knowledge, time predicting task is never considered with DTDG. Explicit time prediction in DTDG learning will be an important research direction in our opinion. For example, in financial stock price prediction, we can model each company in the investment portfolio as a node based on the background of the companies, and then the relation between nodes can be formulated as a graph. In addition to predicting the future price of a particular company's stock, investors are also interested in knowing the time required to hold the stock until the targeted share price is reached. To facilitate explicit time prediction in both supervised and unsupervised DTDG learning, an explicit time DTDG encoder or decoder is required. Future works could be conducted on developing and applying explicit time supervised DTDG learning algorithms to predict the number of snapshots required for a particular change to happen.

Large Scale Dynamic Graph Learning

GNN-based learning models suffer from high memory and CPU power requirement, which becomes more critical in large-scale dynamic graph learning. We observe a recent trend in solving this challenge in static graph learning [41, 5, 24], while there is still large improvement room to address this issue in dynamic graph learning. Optimizing existing dynamic graph learning methods to accommodate large-scale dynamic graphs is a meaningful and interesting future work.

The Significance of Temporal Pattern

As discussed in Sec. 2.4, the evolution of a dynamic network can be described by Eqn (2.1), where the future state of a dynamic graph is correlated to its past. The ultimate goal of dynamic graph learning is to estimate this function and apply it in different graph learning tasks. What if evolution is just a random event that is not correlated with a graph's history? Or more possibly, there is some randomness in the evolution that cannot be explained or learnt. In this case, we believe the performance gain by applying dynamic graph learning methods over static graph learning methods depends on how much randomness is presented in the temporal process. Considering the additional complexity of dynamic graph modelling and learning over static ones, the performance gain via evaluating the significance of temporal patterns in a dynamic graph is very useful for industry practitioners in choosing an appropriate model with low cost.

Implementation Tools for Dynamic Graph Learning

There are Tensorflow [1], Pytorch [66] and Scikit-Learn [67] to help write data pipelines in machine learning algorithm implementations. Some necessary and useful tools, such as DGL [93], Spektral [28], can help implement static graph learning algorithms. However, there are fewer library supports for implementing dynamic graph learning. Future work on creating such convenient tools will be beneficial for both the industry as well as the research community.

2.8 Conclusion

As a data structure, graph data modelling attracts the data science community by its extraordinary expressive power. The world is a dynamic system evolving over time, and so

does the data generated from real-world activities. The dynamic graph is the extension of naive graph modelling to capture this temporal dynamic. Moreover, with the recent development of big data technology and online services, recording and storing graph attributes have become feasible. This urges the need for effective supervised dynamic graph algorithms to perform different machine learning tasks with better accuracy. To provide a comprehensive reference for academia researchers and industry practitioners, this survey is conducted on the following scopes:

- Background of dynamic graph learning with a full-scale systematic summary from storage model, learning purpose, algorithm architecture framework.
- We propose the Three Stages Recurrent Temporal Learning Framework . Based on the proposed temporal learning framework, we discuss how temporal information is learnt by different dynamic graph learning algorithms. Three Stages Recurrent Temporal Learning Framework also provides a general mathematical form of dynamic graph learning algorithms. As far as we know, this is the first and only temporal learning framework which could be applied to both attributed and non-attributed dynamic graph learning.
- Supervised dynamic graph learning algorithms with a detailed introduction from DTDG compatible algorithms to CTDG compatible algorithms, from encoders to decoders and from implicit time algorithms to explicit time algorithms.
- Future research directions according to the topics discussed.

We hope that with this survey paper, we can offer a convenient reference to industry practitioners and facilitate future research for the academic community.

Notation	Description
α	The smoothing factor of EMA
G	A graph
V	Set of nodes $V = v_1, v_2, \cdots v_n$; the input value $\mathbf{V}$ of the attention layer
$V(T)$	Node-wise events observed at period T
E	Set of edges $E = e_1, e_2, \cdots e_n$
$E(T)$	Interaction events observed at period T
v	A node v in V
e	An edge e in E
$e_{i,j}$	the edge start from node i to j
$\mathbf{e}_{ij,t}$	the edge feature vector for edge $e_{i,j}$ at time t
t	Time step / event time
t^-	Time step just before time t
T	Time duration
G_T	Dynamic graph in time duration T
O_T	Observations of a dynamic graph in time duration T
G_t	Dynamic graph at time t
o_t	Observation of a dynamic graph at time t, $o_t \in O_T$
$o_t(v_i)$	A node updating event for node v_i observed at time t
$e_{\{i,j\},t})$	An edge updating event between node v_i and v_j observed at time t
Y	Set of labels in a data set
y_i	A particular label y_i in Y
$\mathbf{X}$	Nodes feature matrix for V
$\mathbf{x}$	Node feature vector in X
w, b, j, φ	Learnable or preset model parameters in scala form
$\mathbf{w}, \mathbf{b}, \mathbf{j}, \varphi$	Learnable or preset model parameters in vector form
$\mathbf{W}, \mathbf{B}, \mathbf{J}$	Learnable or preset model parameters in matrix form
f, λ	Model functions
h, c	Hidden states in scala form
$\mathbf{h}, \mathbf{c}$	Hidden states in vector form
$\mathbf{H}, \mathbf{C}$	Hidden states in matrix form
$\mathbf{h}_{v_i}^k$	Hidden state of node v_i at layer k in a learning model

Table 2.5: List Of Notations

Notation	Description
$N(v_i)$	Neighborhood function which returns the neighboring nodes for the input node v_i
$E(v_i)$	A function returns all edges connecting v_i to its neighbors
$\mathbf{N_t}$	Neighborhood aggregation matrix, formed by stacking each node's neighboring aggregation
$\mathbf{Q}$	The input query of the attention layer
$\mathbf{K}$	The input key of the attention layer; Kalman gain in Kalman filter
$\mathbf{P}$	Positional encoding in TSA; Covariance matrix in Kalman filter
$\mathbf{K^T}$	The superscript T in bold text represents the transpose operation in matrix
$\mathbf{z}_{v_i}$	Embedding vector for node v_i
$\mathbf{Z}$	Graph embedding matrix
$\mathbf{A}$	Adjacency matrix
$\hat{\mathbf{D}}$	Diagonal degree matrix
$\mathbf{L}$	Graph labracian
$\mathbf{m}_{v_i,t}$	Message passed to node v_i at time t
out$(\cdot)$	output function of a learning model
tp$(\cdot)$	Temporal Pattern to learn in Three Stages Recurrent Temporal Learning Framework
asu$(\cdot)$	Attributes Self-Updating function in Three Stages Recurrent Temporal Learning Framework
ap$(\cdot)$	Association Process function Three Stages Recurrent Temporal Learning Framework
mp$(\cdot)$	Message Passing function Three Stages Recurrent Temporal Learning Framework
attn$(\cdot)$	Attention layer
msg$(\cdot)$	message generating function in TGN
mem$(\cdot)$	The memory of a TGN model
agg$(\cdot)$	message aggregating function in TGN
t2v$(\cdot)$	Time2Vec embedding
$\otimes$	Element-wise multiplication
$\|\|$	Concatenation

Table 2.6: List Of Notations

Abbreviation	Description
APAN	Asynchronous Propagation Attention Network
CNN	Convolutional Neural Network
Conv1d	1-d Convolution Neural Network
CTDG	Continuous Time Dynamic Graph
DeepWalk	A graph embedding method based on deep learning and random walk
DNGR	Deep Neural Graph Representation
DTDG	Discrete Time Dynamic Graph
DyGGNN	Dynamic Gated Graph Neural Network
DySAT	Dynamic Self-Attention Network
EvolveGCN	Evolving GCN
EGCU	Evolving GCN unit
EMA	Exponential Moving Average
ARIMA	Auto-Regressive Integrated Moving Average
FNN	Feedforward Neural Network
GCN	Graph Convolution Neural Network
GNN	Graph Neural Network
GraRep	Graph representation with global structural information
GRU	Gated Recurrent Unit
HARP	Hierarchical Representation Learning
HOPE	High Order Proximity Preserved Embedding
Jodie	a coupled recurrent neural network model that learns the embedding trajectories of users
Know-Evolve	Deep evolutionary knowledge network for dynamic knowledge graph learning
LINE	Large-scale Information Network Embedding
LSTM	Long Short Term Memory
M-NMF	Modularized Non-negative Matrix Factorization
RNN	Recurrent Neural Network
SDNE	Structural Deep Network Embedding
softmax	softmax layer
STGCN	Spatial Temporal Graph Convolution Network
TADW	Text Associated DeepWalk
TGN	Temporal Graph Network
TGAT	Temporal Graph Attention Network
TGNN	Temporal Graph Neural Network
TPP	Temporal Point Process
TSA	Temporal Self-Attention

Table 2.7: List Of Abbreviations

Bibliography

[1] Martín Abadi et al. “Tensorflow: A system for large-scale machine learning”. In: *12th {USENIX} symposium on operating systems design and implementation ({OSDI} 16)*. 2016, pp. 265–283.

[2] Amr Ahmed et al. “Distributed large-scale natural graph factorization”. In: *Proceedings of the 22nd international conference on World Wide Web*. 2013, pp. 37–48.

[3] Oriol Artime, José J Ramasco, and Maxi San Miguel. “Dynamics on networks: competition of temporal and topological correlations”. In: *Scientific reports* 7.1 (2017), pp. 1–10.

[4] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. “Neural machine translation by jointly learning to align and translate”. In: *arXiv preprint arXiv:1409.0473* (2014).

[5] Youhui Bai et al. “Efficient Data Loader for Fast Sampling-Based GNN Training on Large Graphs”. In: *IEEE Transactions on Parallel and Distributed Systems* 32.10 (2021), pp. 2541–2556.

[6] Stefan Banach. “Sur les opérations dans les ensembles abstraits et leur application aux équations intégrales”. In: *Fund. math* 3.1 (1922), pp. 133–181.

[7] Nikolaos Bastas et al. “evolve2vec: Learning network representations using temporal unfolding”. In: *International Conference on Multimedia Modeling*. Springer. 2019, pp. 447–458.

[8] Mikhail Belkin and Partha Niyogi. “Laplacian eigenmaps for dimensionality reduction and data representation”. In: *Neural computation* 15.6 (2003), pp. 1373–1396.

[9] Fabricio Benevenuto et al. “Understanding video interactions in youtube”. In: *Proceedings of the 16th ACM international conference on Multimedia*. 2008, pp. 761–764.

[10] Kenneth A Berman. “Vulnerability of scheduled networks and a generalization of Menger’s theorem”. In: *Networks: An International Journal* 28.3 (1996), pp. 125–134.

[11] Toon Bogaerts et al. “A graph CNN-LSTM neural network for short and long-term traffic forecasting based on trajectory data”. In: *Transportation Research Part C: Emerging Technologies* 112 (2020), pp. 62–77.

[12] E Boschee et al. “Integrated Crisis Early Warning System (ICEWS) Coded Event Data”. In: *URL: https://dataverse. harvard. edu/dataverse/icews* (2015).

[13] Shaosheng Cao, Wei Lu, and Qiongkai Xu. “Deep neural networks for learning graph representations”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 30, No.1. 2016.

[14] Shaosheng Cao, Wei Lu, and Qiongkai Xu. “Grarep: Learning graph representations with global structural information”. In: *Proceedings of the 24th ACM international on conference on information and knowledge management*. 2015, pp. 891–900.

[15] Xiaofu Chang et al. “Continuous-Time Dynamic Graph Learning via Neural Interaction Processes”. In: *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*. 2020, pp. 145–154.

[16] Haochen Chen et al. “Harp: Hierarchical representation learning for networks”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 32, No.1. 2018.

[17] Yi-Hsiang Chen and Jen-Tzung Chien. “Continuous-time attention for sequential learning”. In: *Proc. of AAAI Conference on Aritificial Intelligence*. 2021.

[18] Eunjoon Cho, Seth A Myers, and Jure Leskovec. “Friendship and mobility: user movement in location-based social networks”. In: *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*. 2011, pp. 1082–1090.

[19] Peng Cui et al. “A survey on network embedding”. In: *IEEE Transactions on Knowledge and Data Engineering* 31.5 (2018), pp. 833–852.

[20] Sam De Winter et al. “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)*. IEEE. 2018, pp. 1234–1241.

[21] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. “Convolutional neural networks on graphs with fast localized spectral filtering”. In: *Advances in neural information processing systems* 29 (2016), pp. 3844–3852.

[22] Delbert Dueck. *Affinity propagation: clustering data by passing messages*. Citeseer, 2009.

[23] Damien Farine. “The dynamics of transmission and the dynamics of networks”. In: *Journal of Animal Ecology* 86.3 (2017), pp. 415–418.

[24] Claudio Gallicchio and Alessio Micheli. “Fast and deep graph neural networks”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 34, No.04. 2020, pp. 3898–3905.

[25] Matt W Gardner and SR Dorling. "Artificial neural networks (the multilayer perceptron)—a review of applications in the atmospheric sciences". In: *Atmospheric environment* 32.14-15 (1998), pp. 2627–2636.

[26] Justin Gilmer et al. "Neural message passing for quantum chemistry". In: *International conference on machine learning*. PMLR. 2017, pp. 1263–1272.

[27] Justin Gilmer et al. "Neural message passing for quantum chemistry". In: *International Conference on Machine Learning*. PMLR. 2017, pp. 1263–1272.

[28] Daniele Grattarola and Cesare Alippi. "Graph neural networks in tensorflow and keras with spektral". In: *arXiv preprint arXiv:2006.12138* (2020).

[29] Aditya Grover and Jure Leskovec. "node2vec: Scalable feature learning for networks". In: *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*. 2016, pp. 855–864.

[30] İsmail Güneş, Şule Gündüz-Öğüdücü, and Zehra Çataltepe. "Link prediction using time series of neighborhood-based node similarity scores". In: *Data Mining and Knowledge Discovery* 30.1 (2016), pp. 147–180.

[31] William L Hamilton, Rex Ying, and Jure Leskovec. "Inductive representation learning on large graphs". In: *arXiv preprint arXiv:1706.02216* (2017).

[32] William L Hamilton, Rex Ying, and Jure Leskovec. "Representation learning on graphs: Methods and applications". In: *IEEE Data Eng. Bull.* 40.3 (2017).

[33] Zhen Han et al. "Graph hawkes neural network for forecasting on temporal knowledge graphs". In: *arXiv preprint arXiv:2003.13432* (2020).

[34] Alan G Hawkes. "Spectra of some self-exciting and mutually exciting point processes". In: *Biometrika* 58.1 (1971), pp. 83–90.

[35] Sepp Hochreiter and Jürgen Schmidhuber. "Long short-term memory". In: *Neural computation* 9.8 (1997), pp. 1735–1780.

[36] Petter Holme. "Network reachability of real-world contact sequences". In: *Physical Review E* 71.4 (2005), p. 046119.

[39] Zan Huang and Dennis KJ Lin. "The time-series link prediction problem with applications in communication surveillance". In: *INFORMS Journal on Computing* 21.2 (2009), pp. 286–303.

[40] Valerie Isham and Mark Westcott. "A self-correcting point process". In: *Stochastic processes and their applications* 8.3 (1979), pp. 335–347.

[41] Zhihao Jia et al. "Improving the accuracy, scalability, and performance of graph neural networks with roc". In: *Proceedings of Machine Learning and Systems* 2 (2020), pp. 187–198.

[42] Rudolph Emil Kalman et al. "A new approach to linear filtering and prediction problems [J]". In: *Journal of basic Engineering* 82.1 (1960), pp. 35–45.

[43] Seyed Mehran Kazemi et al. "Representation Learning for Dynamic Graphs: A Survey." In: *Journal of Machine Learning Research* 21.70 (2020), pp. 1–73.

[44] Seyed Mehran Kazemi et al. "Time2vec: Learning a vector representation of time". In: *arXiv preprint arXiv:1907.05321* (2019).

[45] David Kempe, Jon Kleinberg, and Amit Kumar. "Connectivity and inference problems for temporal networks". In: *Journal of Computer and System Sciences* 64.4 (2002), pp. 820–842.

[46] Thomas N Kipf and Max Welling. "Semi-supervised classification with graph convolutional networks". In: *arXiv preprint arXiv:1609.02907* (2016).

[47] Thomas N Kipf and Max Welling. "Variational graph auto-encoders". In: *arXiv preprint arXiv:1611.07308* (2016).

[48] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "Imagenet classification with deep convolutional neural networks". In: *Advances in neural information processing systems* 25 (2012), pp. 1097–1105.

[49] Srijan Kumar, Xikun Zhang, and Jure Leskovec. "Learning dynamic embeddings from temporal interactions". In: *arXiv preprint arXiv:1812.02289* (2018).

[50] Srijan Kumar, Xikun Zhang, and Jure Leskovec. "Predicting dynamic embedding trajectory in temporal interaction networks". In: *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 2019, pp. 1269–1278.

[51] Leslie Lamport. "Time, clocks, and the ordering of events in a distributed system". In: *Concurrency: the Works of Leslie Lamport*. Association for Computing Machinery, 2019, pp. 179–196.

[52] Yann LeCun, Yoshua Bengio, et al. "Convolutional networks for images, speech, and time series". In: *The handbook of brain theory and neural networks* 3361.10 (1995), p. 1995.

[53] Jundong Li et al. "Attributed network embedding for learning in a dynamic environment". In: *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*. 2017, pp. 387–396.

[54] Yujia Li et al. "Gated graph sequence neural networks". In: *arXiv preprint arXiv:1511.05493* (2015).

[55] Yao Ma et al. "Streaming graph neural networks". In: *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*. 2020, pp. 719–728.

[56] Franco Manessi, Alessandro Rozza, and Mario Manzo. "Dynamic graph convolutional networks". In: *Pattern Recognition* 97 (2020), p. 107000.

[57] Naoki Masuda and Renaud Lambiotte. *Guide To Temporal Networks, A*. Vol. 6. World Scientific, 2020.

[59] KS Miller, RI Bernstein, and LE Blumenson. "Generalized rayleigh processes". In: *Quarterly of Applied Mathematics* 16.2 (1958), pp. 137–145.

[60] 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.

[61] Apurva Narayan and Peter HO'N Roe. "Learning graph dynamics using deep neural networks". In: *IFAC-PapersOnLine* 51.2 (2018), pp. 433–438.

[62] Giang Hoang Nguyen et al. "Continuous-time dynamic network embeddings". In: *Companion Proceedings of the The Web Conference 2018*. 2018, pp. 969–976.

[63] Mathias Niepert, Mohamed Ahmed, and Konstantin Kutzkov. "Learning convolutional neural networks for graphs". In: *International conference on machine learning*. PMLR. 2016, pp. 2014–2023.

[64] Mingdong Ou et al. "Asymmetric transitivity preserving graph embedding". In: *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*. 2016, pp. 1105–1114.

[65] Aldo Pareja et al. "Evolvegcn: Evolving graph convolutional networks for dynamic graphs". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 34, No.04. 2020, pp. 5363–5370.

[66] Adam Paszke et al. "Pytorch: An imperative style, high-performance deep learning library". In: *Advances in neural information processing systems* 32 (2019), pp. 8026–8037.

[67] Fabian Pedregosa et al. "Scikit-learn: Machine learning in Python". In: *the Journal of machine Learning research* 12 (2011), pp. 2825–2830.

[69] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. "Deepwalk: Online learning of social representations". In: *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. 2014, pp. 701–710.

[70] Trang Pham et al. "Column networks for collective classification". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 31, No.1. 2017.

[71] Jakob Gulddahl Rasmussen. "Temporal point processes: the conditional intensity function". In: *Lecture Notes, Jan* (2011).

[72] Emanuele Rossi et al. "Temporal graph networks for deep learning on dynamic graphs". In: *arXiv preprint arXiv:2006.10637* (2020).

[73] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. "Learning representations by back-propagating errors". In: *nature* 323.6088 (1986), pp. 533–536.

[74] Aravind Sankar et al. "Dysat: Deep neural representation learning on dynamic graphs via self-attention networks". In: *Proceedings of the 13th International Conference on Web Search and Data Mining*. 2020, pp. 519–527.

[75] 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*. PMLR. 2007, pp. 420–427.

[76] Franco Scarselli et al. "The graph neural network model". In: *IEEE transactions on neural networks* 20.1 (2008), pp. 61–80.

[77] Youngjoo Seo et al. "Structured sequence modeling with graph convolutional recurrent networks". In: *International Conference on Neural Information Processing*. Springer. 2018, pp. 362–373.

[78] Paulo Ricardo da Silva Soares and Ricardo Bastos Cavalcante Prudêncio. "Time series based link prediction". In: *The 2012 international joint conference on neural networks (IJCNN)*. IEEE. 2012, pp. 1–7.

[79] Joakim Skardinga, Bogdan Gabrys, and Katarzyna Musial. "Foundations and modelling of dynamic networks using dynamic graph neural networks: A survey". In: *IEEE Access* (2021).

[80] Johan AK Suykens and Joos Vandewalle. "Least squares support vector machine classifiers". In: *Neural processing letters* 9.3 (1999), pp. 293–300.

[81] Aynaz Taheri, Kevin Gimpel, and Tanya Berger-Wolf. "Learning to represent the evolution of dynamic graphs with recurrent models". In: *Companion Proceedings of The 2019 World Wide Web Conference*. 2019, pp. 301–307.

[82] Jian Tang et al. "Line: Large-scale information network embedding". In: *Proceedings of the 24th international conference on world wide web*. 2015, pp. 1067–1077.

[84] Riitta Toivonen et al. "A comparative study of social network models: Network evolution models and nodal attribute models". In: *Social networks* 31.4 (2009), pp. 240–254.

[85] Rakshit Trivedi et al. “Dyrep: Learning representations over dynamic graphs”. In: *International conference on learning representations*. 2019.

[86] Rakshit Trivedi et al. “Know-evolve: Deep temporal reasoning for dynamic knowledge graphs”. In: *international conference on machine learning*. PMLR. 2017, pp. 3462–3471.

[87] Ashish Vaswani et al. “Attention is all you need”. In: *Advances in neural information processing systems* 30 (2017).

[88] Petar Veličković et al. “Graph attention networks”. In: *arXiv preprint arXiv:1710.10903* (2017).

[89] Chengxin Wang, Shaofeng Cai, and Gary Tan. “Graphtcn: Spatio-temporal interaction modeling for human trajectory prediction”. In: *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*. Waikoloa, HI, USA: IEEE, 2021, pp. 3450–3459.

[90] Daixin Wang, Peng Cui, and Wenwu Zhu. “Structural deep network embedding”. In: *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*. 2016, pp. 1225–1234.

[91] Kaijun Wang et al. “Adaptive affinity propagation clustering”. In: *arXiv preprint arXiv:0805.1096* (2008).

[93] Minjie Wang et al. “Deep Graph Library: Towards Efficient and Scalable Deep Learning on Graphs.” In: (2019).

[94] Xiao Wang et al. “Community preserving network embedding”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 31, No.1. 2017.

[95] Xuhong Wang et al. "APAN: Asynchronous Propagation Attention Network for Real-time Temporal Graph Embedding". In: *Proceedings of the 2021 International Conference on Management of Data*. 2021, pp. 2628–2638.

[96] Jesse Weaver and Paul Tarjan. "Facebook linked data via the graph API". In: *Semantic Web* 4.3 (2013), pp. 245–250.

[97] Zonghan Wu et al. "A comprehensive survey on graph neural networks". In: *IEEE transactions on neural networks and learning systems* (2020).

[98] Bing Xu et al. "Empirical evaluation of rectified activations in convolutional network". In: *arXiv preprint arXiv:1505.00853* (2015).

[99] Da Xu et al. "Inductive representation learning on temporal graphs". In: *arXiv preprint arXiv:2002.07962* (2020).

[100] Da Xu et al. *Self-attention with functional time representation learning*. 2019. arXiv: 1911.12864.

[101] B Bui Xuan, Afonso Ferreira, and Aubin Jarry. "Computing shortest, fastest, and foremost journeys in dynamic networks". In: *International Journal of Foundations of Computer Science* 14.02 (2003), pp. 267–285.

[102] Sijie Yan, Yuanjun Xiong, and Dahua Lin. "Spatial temporal graph convolutional networks for skeleton-based action recognition". In: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 32, No.1. 2018.

[103] Cheng Yang et al. "Network representation learning with rich text information". In: *Twenty-fourth international joint conference on artificial intelligence*. 2015.

[104] Yuan Yuan et al. "Temporal dynamic graph lstm for action-driven video object detection". In: *Proceedings of the IEEE international conference on computer vision*. 2017, pp. 1801–1810.

[105] Aya Zaki et al. "Comprehensive Survey on Dynamic Graph Models". In: *International Journal of Advanced Computer Science and Applications* 7.2 (2016). ISSN: 2158107X. DOI: 10.14569/ijacsa.2016.070273.

[106] Li Zheng et al. "AddGraph: Anomaly Detection in Dynamic Graph Using Attention-based Temporal GCN." In: *IJCAI*. 2019, pp. 4419–4425.

[107] Jie Zhou et al. "Graph neural networks: A review of methods and applications". In: *AI Open* 1 (2020), pp. 57–81.

Chapter 3

Discussion and Future Works

3.1 Preliminary Study of Measuring the Significance of Temporal Pattern

Because of their more complex structure, dynamic graph learning algorithms usually require more computing resources than those for static graphs. When deciding whether we should use a dynamic graph algorithm for a particular graph analysis task, the following two questions help us to make the decision:

- Can we model the given graph as a dynamic graph?
- Is the performance gain offered by a dynamic graph learning algorithm worth the extra compute resources it requires?

If the answer to these two questions is Yes, then we should use a dynamic graph learning algorithm. While the first question is straightforward to answer, the second one is not. We can answer the second question by experimenting with different dynamic and static graph learning algorithms and comparing their performance. But this costs considerable time and

compute resources. As discussed in manuscript two presented in chapter 2, if the temporal pattern is significant in a dynamic graph's evolution, then analyzing it with a dynamic graph learning algorithm may offer better performance because it can learn the temporal information. It would have a great impact if we could estimate the performance gain by just evaluating the significance of the temporal pattern presented in the input graph. To start tackling this research question, I conduct a preliminary study on measuring the significance of temporal pattern for DTDG in this section.

We experiment with the Temporal Correlation Coefficient [83] and observe that it over estimates the significance of temporal pattern for the DTDGs that have isolated nodes. We also develop a more efficient and robust measure named Normalized Jaccard Similarity to estimate the significance of temporal pattern for DTDGs.

3.1.1 Estimate the Significance of Temporal Pattern

For a DTDG, if a function exists as described in Eqn. 2.1 that we can use to predict its future without errors, then the degree of significance of its temporal pattern is 100%. However, in reality, we better assume there is a random error ϵ_e for edges prediction and ϵ_x for nodes feature prediction:

$$(E_{t_{n+1}}, X_{t_{n+1}}, t_{n+1}) = (\hat{E}_{t_{n+1}} + \epsilon_{e,t_{n+1}}, \hat{X}_{t_{n+1}} + \epsilon_{x,t_{n+1}}, t_{n+1}) \tag{3.1}$$

And the significance of temporal pattern can be defined as:

$$C = (1 - \text{mean}(\frac{\epsilon_{e,t}}{\sqrt{|E_{t-1}||E_t|}})) * (1 - \text{mean}(\frac{\epsilon_{x,t}}{\sqrt{|\mathbf{X}_{t-1}||\mathbf{X}_t|}})) \tag{3.2}$$

where mean($\frac{\epsilon_{e,t}}{\sqrt{|E_{t-1}||E_t|}}$) is the mean value of the normalized random error $\epsilon_{e,t}$; and mean($\frac{\epsilon_{x,t}}{\sqrt{|\mathbf{X}_{t-1}||\mathbf{X}_t|}}$) is the mean value of the normalized random error $\epsilon_{x,t}$. $C \in [0, 1]$ and the bigger it is, the more significant the temporal pattern presented in the particular dynamic graph. In this preliminary study, we focus on estimating $\epsilon_{e,t}$ for edge prediction, assuming the node features do not change over time and hence $\epsilon_{x,t}$ is always 0, which is the case of a citation graph. Such that we have

$$C = (1 - \text{mean}(\frac{\epsilon_{e,t}}{\sqrt{|E_{t-1}||E_t|}})) * (1 - 0) = 1 - \text{mean}(\frac{\epsilon_{e,t}}{\sqrt{|E_{t-1}||E_t|}}) \tag{3.3}$$

3.1.2 Temporal Correlation Coefficient

In equation 3.3, $\epsilon_{e,t}$ depends on Eqn. 2.3b which we don't know yet. To approximate $\epsilon_{e,t}$ we can assume the identity function for Eqn. 2.3b and thus we can use the Temporal Correlation Coefficient [83] to estimate C for DTDG that has node features that never change:

$$C = \frac{1}{N(t_{max} - 1)} \sum_{t=1}^{t_{max}-1} \sum_{i=1}^{N} \frac{\sum_{j=1}^{N} A_{ij}(t) A_{ij}(t+1)}{\sqrt{[\sum_{j=1}^{N} A_{ij}(t)][\sum_{j=1}^{N} A_{ij}(t+1)]}} \tag{3.4}$$

The input to calculate the Temporal Correlation Coefficient is the adjacency matrix of all snapshots in a DTDG. Assuming the given DTDG has N nodes and no changes to its nodes set, its time complexity is $O(t_{max}N^2)$. It is inefficient for sparse dynamic graphs.

Moreover, as will be shown by the experiment, the Temporal Correlation Coefficient tends to report a high estimate of the significance of the temporal pattern for DTDG with many isolated nodes. To address these two disadvantages, we propose the Normalized Jaccard Similarity, which is more efficient and robust.

3.1.3 Normalized Jaccard Similarity

With the same assumption that Eqn. 2.3b is the identity function, We develop a measure to estimate the significance of temporal pattern based on the Jaccard Similarity of the given DTDG's edges set. The proposed measure is named Normalized Jaccard Similarity and is described in the following equation:

$$C = \frac{1}{t_{max} - 1)} \sum_{t=1}^{t_{max}-1} \frac{|E_t \cap E_{t+1}|}{|E_t \cup E_{t+1}|} \tag{3.5}$$

where the operation $|A|$ represents the size of the set A. The time complexity of the Normalized Jaccard Similarity is $O(t_{max}E)$ where E is the average number of edges in all snapshots. Since the maximum of E is N^2, which only happens in fully connected graphs, we have $E \leq N^2 \implies O(t_{max}E)O(t_{max}N^2)$. In most cases, the Normalized Jaccard Similarity is more efficient than the Temporal Correlation Coefficient, especially for sparse dynamic graphs such as social graphs. And in the worse cases, the Normalized Jaccard Similarity has the same time complexity as the Temporal Correlation Coefficient.

3.1.4 Experiments

To validate the proposed measures, we conducted an experiment to calculate the Temporal Correlation Coefficient and the Normalized Jaccard Similarity for the ogbn-arxiv citation graph [37], a dynamic graph that has unchanged node features, and the r8-text-classification dataset [38], a collection of static graphs generated from different documents.

The ogbn-arxiv dataset models the citation network between all Computer Science (CS) arXiv papers. Each edge represents one citation. The node feature is represented as a 128-dimensional vector. This vector is the average of the words embedding encoding the title and abstract. Each word embedding is computed by the skip-gram model [58] trained on

the Microsoft Academic Graph corpus [92]. The ogbn-arxiv citation network has 169,343 nodes, 1,166,243 edges and 35 snapshots.

The r8-text-classification dataset is a subset of the Reuters 21578 datasets. It has 7674 documents. Each document is modelled as a graph of words with the same method as in Huang et el. [38]. The node feature is the pre-trained GloVe word embedding [68]. Each document mimics a snapshot in a DTDG. They are randomly ordered to simulate a dynamic graph without any temporal patterns presented.

Because the node feature in both datasets does not change across different snapshots, they fulfill the assumption in the study above.

The Temporal Correlation Coefficient and the Normalized Jaccard Similarity of the two datasets are summarized in table 3.1.

dataset	Temporal Correlation Coefficient	Normalized Jaccard Similarity
ogbn-arxiv	0.884	0.635
r8-text-classification	0.996	0.034

Table 3.1: Experiment Result

From the experiment, we discover that, though the Temporal Correlation Coefficient reported a high estimate of 0.884 for the ogbn-arxiv dataset, it also gave out a high estimate of 0.996 for the dataset r8-text-classification, which should have no temporal pattern presented. If a node is isolated in two consecutive snapshots, its Temporal Correlation Coefficient would be 1. Because text graphs have a very sparse network structure, many nodes in the graphs are isolated nodes with only self-connection. Therefore the Temporal Correlation Coefficient reports high estimates for DTDG with many isolated nodes.

On the other hand, the Normalized Jaccard Similarity provided a relatively accurate estimate for both datasets. It reported a high estimate of 0.635 for the ogbn-arxiv dataset and a low estimate of 0.034 for the r8-text-classification dataset. The Normalized Jaccard

Similarity is more robust than the Temporal Correlation Coefficient in estimating the significance of temporal pattern for DTDG.

3.2 Future Works

3.2.1 Evaluate The Significance of Temporal Pattern

In chapter 2, I developed Three Stages Recurrent Temporal Learning Framework to explain how to learn temporal information from the input graph's evolution. And I also discussed that the performance gain of dynamic graph learning algorithms over static ones might depend on the significance of the temporal pattern in the given graph's history. Considering the additional complexity of dynamic graph modelling and learning over the static ones, predicting the performance gain via evaluating the significance of temporal pattern presented in a dynamic graph is very useful for the practitioners to choose an appropriate model with low cost.

Based on Three Stages Recurrent Temporal Learning Framework , I conducted a preliminary study on measuring the significance of the temporal patterns for DTDG in the section above. In the future, I will extend this initial study to validate the relationship between the significance of the temporal pattern and the performance gain with a dynamic graph learning model over a static one. Another related future work is to extend this research to CTDG as well.

3.2.2 From a Taxonomy Concept to a Real Tool Box

In my literature review of the recent development of dynamic graph learning algorithms in chapter 2, I developed a taxonomy to classify the different algorithms. The first level

in this taxonomy is the application target, whether it is for DTDG or CTDG. The second taxonomy level is the temporal learning schema, which includes implicit and explicit time learning models. This taxonomy summarizes how different learning algorithms learn the temporal information and serves as a structural foundation to build a library to offer all reviewed dynamic graph learning algorithms to the practitioners. The proposed taxonomy inspires one of my future works: to build a software library named SpiroGraph.

SpiroGraph is an ongoing work which aims to help practitioners with limited machine learning knowledge to build a cloud naive machine learning pipeline for data that fits the dynamic graph modelling. It is similar to DGL [93] and Spektral [28] which specialized in static graph learning algorithms. I believe SpiroGraph will be a handy tool for the industry and the research community. It will be structured similarly to the taxonomy presented in chapter 2. Users can hence easily find the right tool based on their requirements.

3.2.3 Synthetic Dynamic Graphs

Machine learning with dynamic graphs is an emerging research topic. To provide the community with a handy dataset to test different dynamic graph learning algorithms, I plan to develop a synthetic dynamic graph generating algorithm and tool as part of SpiroGraph based on the Three Stages Recurrent Temporal Learning Framework proposed in chapter 2.

Chapter 4

Conclusion

Dynamic graph learning is becoming a hot topic in the research community and the industry. To understand more about dynamic graph learning and how a learning algorithm learns the temporal information, I conducted the following research works in this thesis:

- I summarized the recent development in supervised dynamic graph learning and proposed a temporal learning framework in manuscript one presented in chapter 2.
- At 3, I did a preliminary study on measuring the significance of temporal patterns for DTDG.

The above works taught me how different dynamic graph learning algorithms learn temporal information. Based on what has been learnt, I developed a general mathematical definition of the significance of the temporal pattern. I also proposed the Normalized Jaccard Similarity to measure it for DTDGs with no changes in their node features.

Bibliography

[1] Martín Abadi et al. “Tensorflow: A system for large-scale machine learning”. In: *12th {USENIX} symposium on operating systems design and implementation ({OSDI} 16)*. 2016, pp. 265–283.

[2] Amr Ahmed et al. “Distributed large-scale natural graph factorization”. In: *Proceedings of the 22nd international conference on World Wide Web*. 2013, pp. 37–48.

[3] Oriol Artime, José J Ramasco, and Maxi San Miguel. “Dynamics on networks: competition of temporal and topological correlations”. In: *Scientific reports* 7.1 (2017), pp. 1–10.

[4] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. “Neural machine translation by jointly learning to align and translate”. In: *arXiv preprint arXiv:1409.0473* (2014).

[5] Youhui Bai et al. “Efficient Data Loader for Fast Sampling-Based GNN Training on Large Graphs”. In: *IEEE Transactions on Parallel and Distributed Systems* 32.10 (2021), pp. 2541–2556.

[6] Stefan Banach. “Sur les opérations dans les ensembles abstraits et leur application aux équations intégrales”. In: *Fund. math* 3.1 (1922), pp. 133–181.

[7] Nikolaos Bastas et al. “evolve2vec: Learning network representations using temporal unfolding”. In: *International Conference on Multimedia Modeling*. Springer. 2019, pp. 447–458.

[8] Mikhail Belkin and Partha Niyogi. “Laplacian eigenmaps for dimensionality reduction and data representation”. In: *Neural computation* 15.6 (2003), pp. 1373–1396.

[9] Fabricio Benevenuto et al. “Understanding video interactions in youtube”. In: *Proceedings of the 16th ACM international conference on Multimedia*. 2008, pp. 761–764.

[10] Kenneth A Berman. “Vulnerability of scheduled networks and a generalization of Menger’s theorem”. In: *Networks: An International Journal* 28.3 (1996), pp. 125–134.

[11] Toon Bogaerts et al. “A graph CNN-LSTM neural network for short and long-term traffic forecasting based on trajectory data”. In: *Transportation Research Part C: Emerging Technologies* 112 (2020), pp. 62–77.

[12] E Boschee et al. “Integrated Crisis Early Warning System (ICEWS) Coded Event Data”. In: *URL: https://dataverse. harvard. edu/dataverse/icews* (2015).

[13] Shaosheng Cao, Wei Lu, and Qiongkai Xu. “Deep neural networks for learning graph representations”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 30, No.1. 2016.

[14] Shaosheng Cao, Wei Lu, and Qiongkai Xu. “Grarep: Learning graph representations with global structural information”. In: *Proceedings of the 24th ACM international on conference on information and knowledge management*. 2015, pp. 891–900.

[15] Xiaofu Chang et al. “Continuous-Time Dynamic Graph Learning via Neural Interaction Processes”. In: *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*. 2020, pp. 145–154.

[16] Haochen Chen et al. “Harp: Hierarchical representation learning for networks”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 32, No.1. 2018.

[17] Yi-Hsiang Chen and Jen-Tzung Chien. “Continuous-time attention for sequential learning”. In: *Proc. of AAAI Conference on Aritificial Intelligence*. 2021.

[18] Eunjoon Cho, Seth A Myers, and Jure Leskovec. “Friendship and mobility: user movement in location-based social networks”. In: *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*. 2011, pp. 1082–1090.

[19] Peng Cui et al. “A survey on network embedding”. In: *IEEE Transactions on Knowledge and Data Engineering* 31.5 (2018), pp. 833–852.

[20] Sam De Winter et al. “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)*. IEEE. 2018, pp. 1234–1241.

[21] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. “Convolutional neural networks on graphs with fast localized spectral filtering”. In: *Advances in neural information processing systems* 29 (2016), pp. 3844–3852.

[22] Delbert Dueck. *Affinity propagation: clustering data by passing messages*. Citeseer, 2009.

[23] Damien Farine. “The dynamics of transmission and the dynamics of networks”. In: *Journal of Animal Ecology* 86.3 (2017), pp. 415–418.

[24] Claudio Gallicchio and Alessio Micheli. “Fast and deep graph neural networks”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 34, No.04. 2020, pp. 3898–3905.

[25] Matt W Gardner and SR Dorling. "Artificial neural networks (the multilayer perceptron)—a review of applications in the atmospheric sciences". In: *Atmospheric environment* 32.14-15 (1998), pp. 2627–2636.

[26] Justin Gilmer et al. "Neural message passing for quantum chemistry". In: *International conference on machine learning*. PMLR. 2017, pp. 1263–1272.

[27] Justin Gilmer et al. "Neural message passing for quantum chemistry". In: *International Conference on Machine Learning*. PMLR. 2017, pp. 1263–1272.

[28] Daniele Grattarola and Cesare Alippi. "Graph neural networks in tensorflow and keras with spektral". In: *arXiv preprint arXiv:2006.12138* (2020).

[29] Aditya Grover and Jure Leskovec. "node2vec: Scalable feature learning for networks". In: *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*. 2016, pp. 855–864.

[30] İsmail Güneş, Şule Gündüz-Öğüdücü, and Zehra Çataltepe. "Link prediction using time series of neighborhood-based node similarity scores". In: *Data Mining and Knowledge Discovery* 30.1 (2016), pp. 147–180.

[31] William L Hamilton, Rex Ying, and Jure Leskovec. "Inductive representation learning on large graphs". In: *arXiv preprint arXiv:1706.02216* (2017).

[32] William L Hamilton, Rex Ying, and Jure Leskovec. "Representation learning on graphs: Methods and applications". In: *IEEE Data Eng. Bull.* 40.3 (2017).

[33] Zhen Han et al. "Graph hawkes neural network for forecasting on temporal knowledge graphs". In: *arXiv preprint arXiv:2003.13432* (2020).

[34] Alan G Hawkes. "Spectra of some self-exciting and mutually exciting point processes". In: *Biometrika* 58.1 (1971), pp. 83–90.

[35] Sepp Hochreiter and Jürgen Schmidhuber. "Long short-term memory". In: *Neural computation* 9.8 (1997), pp. 1735–1780.

[36] Petter Holme. "Network reachability of real-world contact sequences". In: *Physical Review E* 71.4 (2005), p. 046119.

[37] Weihua Hu et al. "Open graph benchmark: Datasets for machine learning on graphs". In: *Advances in neural information processing systems* 33 (2020), pp. 22118–22133.

[38] Lianzhe Huang et al. "Text level graph neural network for text classification". In: *arXiv preprint arXiv:1910.02356* (2019).

[39] Zan Huang and Dennis KJ Lin. "The time-series link prediction problem with applications in communication surveillance". In: *INFORMS Journal on Computing* 21.2 (2009), pp. 286–303.

[40] Valerie Isham and Mark Westcott. "A self-correcting point process". In: *Stochastic processes and their applications* 8.3 (1979), pp. 335–347.

[41] Zhihao Jia et al. "Improving the accuracy, scalability, and performance of graph neural networks with roc". In: *Proceedings of Machine Learning and Systems* 2 (2020), pp. 187–198.

[42] Rudolph Emil Kalman et al. "A new approach to linear filtering and prediction problems [J]". In: *Journal of basic Engineering* 82.1 (1960), pp. 35–45.

[43] Seyed Mehran Kazemi et al. "Representation Learning for Dynamic Graphs: A Survey." In: *Journal of Machine Learning Research* 21.70 (2020), pp. 1–73.

[44] Seyed Mehran Kazemi et al. "Time2vec: Learning a vector representation of time". In: *arXiv preprint arXiv:1907.05321* (2019).

[45] David Kempe, Jon Kleinberg, and Amit Kumar. "Connectivity and inference problems for temporal networks". In: *Journal of Computer and System Sciences* 64.4 (2002), pp. 820–842.

[46] Thomas N Kipf and Max Welling. "Semi-supervised classification with graph convolutional networks". In: *arXiv preprint arXiv:1609.02907* (2016).

[47] Thomas N Kipf and Max Welling. "Variational graph auto-encoders". In: *arXiv preprint arXiv:1611.07308* (2016).

[48] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "Imagenet classification with deep convolutional neural networks". In: *Advances in neural information processing systems* 25 (2012), pp. 1097–1105.

[49] Srijan Kumar, Xikun Zhang, and Jure Leskovec. "Learning dynamic embeddings from temporal interactions". In: *arXiv preprint arXiv:1812.02289* (2018).

[50] Srijan Kumar, Xikun Zhang, and Jure Leskovec. "Predicting dynamic embedding trajectory in temporal interaction networks". In: *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 2019, pp. 1269–1278.

[51] Leslie Lamport. "Time, clocks, and the ordering of events in a distributed system". In: *Concurrency: the Works of Leslie Lamport*. Association for Computing Machinery, 2019, pp. 179–196.

[52] Yann LeCun, Yoshua Bengio, et al. "Convolutional networks for images, speech, and time series". In: *The handbook of brain theory and neural networks* 3361.10 (1995), p. 1995.

[53] Jundong Li et al. "Attributed network embedding for learning in a dynamic environment". In: *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*. 2017, pp. 387–396.

[54] Yujia Li et al. "Gated graph sequence neural networks". In: *arXiv preprint arXiv:1511.05493* (2015).

[55] Yao Ma et al. "Streaming graph neural networks". In: *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*. 2020, pp. 719–728.

[56] Franco Manessi, Alessandro Rozza, and Mario Manzo. "Dynamic graph convolutional networks". In: *Pattern Recognition* 97 (2020), p. 107000.

[57] Naoki Masuda and Renaud Lambiotte. *Guide To Temporal Networks, A*. Vol. 6. World Scientific, 2020.

[58] Tomas Mikolov et al. "Distributed representations of words and phrases and their compositionality". In: *Advances in neural information processing systems* 26 (2013).

[59] KS Miller, RI Bernstein, and LE Blumenson. "Generalized rayleigh processes". In: *Quarterly of Applied Mathematics* 16.2 (1958), pp. 137–145.

[60] 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.

[61] Apurva Narayan and Peter HO'N Roe. "Learning graph dynamics using deep neural networks". In: *IFAC-PapersOnLine* 51.2 (2018), pp. 433–438.

[62] Giang Hoang Nguyen et al. "Continuous-time dynamic network embeddings". In: *Companion Proceedings of the The Web Conference 2018*. 2018, pp. 969–976.

[63] Mathias Niepert, Mohamed Ahmed, and Konstantin Kutzkov. "Learning convolutional neural networks for graphs". In: *International conference on machine learning*. PMLR. 2016, pp. 2014–2023.

[64] Mingdong Ou et al. "Asymmetric transitivity preserving graph embedding". In: *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*. 2016, pp. 1105–1114.

[65] Aldo Pareja et al. "Evolvegcn: Evolving graph convolutional networks for dynamic graphs". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 34, No.04. 2020, pp. 5363–5370.

[66] Adam Paszke et al. "Pytorch: An imperative style, high-performance deep learning library". In: *Advances in neural information processing systems* 32 (2019), pp. 8026–8037.

[67] Fabian Pedregosa et al. "Scikit-learn: Machine learning in Python". In: *the Journal of machine Learning research* 12 (2011), pp. 2825–2830.

[68] Jeffrey Pennington, Richard Socher, and Christopher D Manning. "Glove: Global vectors for word representation". In: *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*. 2014, pp. 1532–1543.

[69] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. "Deepwalk: Online learning of social representations". In: *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. 2014, pp. 701–710.

[70] Trang Pham et al. "Column networks for collective classification". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 31, No.1. 2017.

[71] Jakob Gulddahl Rasmussen. "Temporal point processes: the conditional intensity function". In: *Lecture Notes, Jan* (2011).

[72] Emanuele Rossi et al. “Temporal graph networks for deep learning on dynamic graphs”. In: *arXiv preprint arXiv:2006.10637* (2020).

[73] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. “Learning representations by back-propagating errors”. In: *nature* 323.6088 (1986), pp. 533–536.

[74] Aravind Sankar et al. “Dysat: Deep neural representation learning on dynamic graphs via self-attention networks”. In: *Proceedings of the 13th International Conference on Web Search and Data Mining*. 2020, pp. 519–527.

[75] 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*. PMLR. 2007, pp. 420–427.

[76] Franco Scarselli et al. “The graph neural network model”. In: *IEEE transactions on neural networks* 20.1 (2008), pp. 61–80.

[77] Youngjoo Seo et al. “Structured sequence modeling with graph convolutional recurrent networks”. In: *International Conference on Neural Information Processing*. Springer. 2018, pp. 362–373.

[78] Paulo Ricardo da Silva Soares and Ricardo Bastos Cavalcante Prudêncio. “Time series based link prediction”. In: *The 2012 international joint conference on neural networks (IJCNN)*. IEEE. 2012, pp. 1–7.

[79] Joakim Skardinga, Bogdan Gabrys, and Katarzyna Musial. “Foundations and modelling of dynamic networks using dynamic graph neural networks: A survey”. In: *IEEE Access* (2021).

[80] Johan AK Suykens and Joos Vandewalle. “Least squares support vector machine classifiers”. In: *Neural processing letters* 9.3 (1999), pp. 293–300.

[81] Aynaz Taheri, Kevin Gimpel, and Tanya Berger-Wolf. "Learning to represent the evolution of dynamic graphs with recurrent models". In: *Companion Proceedings of The 2019 World Wide Web Conference*. 2019, pp. 301–307.

[82] Jian Tang et al. "Line: Large-scale information network embedding". In: *Proceedings of the 24th international conference on world wide web*. 2015, pp. 1067–1077.

[83] John Tang et al. "Small-world behavior in time-varying graphs". In: *Physical Review E* 81.5 (2010), p. 055101.

[84] Riitta Toivonen et al. "A comparative study of social network models: Network evolution models and nodal attribute models". In: *Social networks* 31.4 (2009), pp. 240–254.

[85] Rakshit Trivedi et al. "Dyrep: Learning representations over dynamic graphs". In: *International conference on learning representations*. 2019.

[86] Rakshit Trivedi et al. "Know-evolve: Deep temporal reasoning for dynamic knowledge graphs". In: *international conference on machine learning*. PMLR. 2017, pp. 3462–3471.

[87] Ashish Vaswani et al. "Attention is all you need". In: *Advances in neural information processing systems* 30 (2017).

[88] Petar Veličković et al. "Graph attention networks". In: *arXiv preprint arXiv:1710.10903* (2017).

[89] Chengxin Wang, Shaofeng Cai, and Gary Tan. "Graphtcn: Spatio-temporal interaction modeling for human trajectory prediction". In: *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*. Waikoloa, HI, USA: IEEE, 2021, pp. 3450–3459.

[90] Daixin Wang, Peng Cui, and Wenwu Zhu. "Structural deep network embedding". In: *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*. 2016, pp. 1225–1234.

[91] Kaijun Wang et al. "Adaptive affinity propagation clustering". In: *arXiv preprint arXiv:0805.1096* (2008).

[92] Kuansan Wang et al. "Microsoft academic graph: When experts are not enough". In: *Quantitative Science Studies* 1.1 (2020), pp. 396–413.

[93] Minjie Wang et al. "Deep Graph Library: Towards Efficient and Scalable Deep Learning on Graphs." In: (2019).

[94] Xiao Wang et al. "Community preserving network embedding". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 31, No.1. 2017.

[95] Xuhong Wang et al. "APAN: Asynchronous Propagation Attention Network for Real-time Temporal Graph Embedding". In: *Proceedings of the 2021 International Conference on Management of Data*. 2021, pp. 2628–2638.

[96] Jesse Weaver and Paul Tarjan. "Facebook linked data via the graph API". In: *Semantic Web* 4.3 (2013), pp. 245–250.

[97] Zonghan Wu et al. "A comprehensive survey on graph neural networks". In: *IEEE transactions on neural networks and learning systems* (2020).

[98] Bing Xu et al. "Empirical evaluation of rectified activations in convolutional network". In: *arXiv preprint arXiv:1505.00853* (2015).

[99] Da Xu et al. "Inductive representation learning on temporal graphs". In: *arXiv preprint arXiv:2002.07962* (2020).

[100] Da Xu et al. *Self-attention with functional time representation learning*. 2019. arXiv: 1911.12864.

[101] B Bui Xuan, Afonso Ferreira, and Aubin Jarry. "Computing shortest, fastest, and foremost journeys in dynamic networks". In: *International Journal of Foundations of Computer Science* 14.02 (2003), pp. 267–285.

[102] Sijie Yan, Yuanjun Xiong, and Dahua Lin. "Spatial temporal graph convolutional networks for skeleton-based action recognition". In: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 32, No.1. 2018.

[103] Cheng Yang et al. "Network representation learning with rich text information". In: *Twenty-fourth international joint conference on artificial intelligence*. 2015.

[104] Yuan Yuan et al. "Temporal dynamic graph lstm for action-driven video object detection". In: *Proceedings of the IEEE international conference on computer vision*. 2017, pp. 1801–1810.

[105] Aya Zaki et al. "Comprehensive Survey on Dynamic Graph Models". In: *International Journal of Advanced Computer Science and Applications* 7.2 (2016). ISSN: 2158107X. DOI: 10.14569/ijacsa.2016.070273.

[106] Li Zheng et al. "AddGraph: Anomaly Detection in Dynamic Graph Using Attention-based Temporal GCN." In: *IJCAI*. 2019, pp. 4419–4425.

[107] Jie Zhou et al. "Graph neural networks: A review of methods and applications". In: *AI Open* 1 (2020), pp. 57–81.