Link Prediction on CV Graphs: A temporal Graph Neural Network approach. (In collaboration with Euranova)

Nima Farnoodian

June 2022

Abstract

This thesis studies one of the recent advancements on the graph and its application in terms of the power of predictability that they can endow. More clearly, Machine Learning on Graphs has recently drawn much attention as it has enabled us to perform Graph Classification, Node Recommendation, Node Classification, Node Clustering, and link prediction. Following an agreement with our industrial partner, **Euranova**, this thesis was assigned to scrutinize machine learning on dynamic networks and examine the popular *Temporal Graph Network — a deep learning framework for dynamic graphs*—, aka TGN, with a challenging curriculum-vita dataset with thousand applicants provided by Euranova. In case we mine the dataset and obtain a dynamic bipartite CV graph consisting of thousand nodes of distinct types (applicants and roles), the aim is to see how sufficiently TGN can help build a role predictor that could also be deemed a role recommender. This curiosity comes from the fact that people may change their roles over time as they get more experienced, educated, or seniority. For example, a junior data scientist will become a senior data scientist after several years, or a developer may become Development Lead as s/he acquires sufficient knowledge and experience. Thus, the time dimension defines and captures the evolution, and avoiding time in any models predicting the roles or jobs disregards this verity.

In light of this reality, this thesis presents a role predictor and top-k recommendation system based on the provided CV Data and TGN model that can record the evolutionary patterns and recommend subsequent roles for applicants based on their experiences and skills. The experimental findings demonstrate that the proposed model is able to recommend k roles out of thousands of available roles such that the predicted roles closely resemble the ground-truth roles.

Keywords— Temporal Graph Network, Dynamic Graph, Link Prediction, Role Prediction. Top-k Role Recommendation

Contents

List of Figures

List of Tables

Acknowledgements

"I shall be telling this with a smile Somewhere ages and ages hence: Two roads diverged in a wood, and I, I took the one less traveled by, And that has opened my eyes to the world"

And finally, it came, the last days of my journey. The journey that I started two years ago, but its dream took shape when I was an 11-year-old boy and awarded a Pentium III computer taking me to the world full of adventures and excitements called "Computer Science". My journey was full of many good and bad memories shared with many people.

My deepest gratitude must go to my supervisor, Dr. Siegfried Nijssen, not only for all his unfailing advice, support, and encouragement but also for his trust, kindness, and patience. He has a way of motivating his students towards continual improvement.

Appreciation is extended to Dr. Gianmarco Aversano, my second supervisor from Euranova, for his kindness, guidance, and thoughtful advice. His great help and support will not be forgotten.

It has been my honor to accomplish my thesis in collaboration with Euranova. I would also like to express my sincere gratitude to Dr. Madalina Ciortan from Euranova for her unlimited assistance.

My friends were sources of support and joy during these years. The friendships we have developed will last a lifetime. Special thanks go to Dr. Ali Kotlar Mehmandost, Andia Danlée Fathi, Dr. Yann Danlée, Bastien Michaux-Bellomo, Carla Barkat, Ali Alizade Eslami, and his wife, Nasim.

A big appreciation from the bottom of my heart goes to my wife, Atefeh Bahrami, whose unconditional love was the biggest source of energy in my journey.

Words cannot express my gratitude to my family. My parents, Farangis and Farajollah. I owe them my whole life. Their love, prayer, and sacrifice are my driving force. I wish I could show them how I am thankful for having them. My wonderful siblings, Mitra and Reza, have wholeheartedly supported me and are two shining stars of my life whose glitters keep my life bright.

My nephew, Sam, a wonderful reminder of all that makes life wonderful.

And in the end, I should say that every single decision made in life comes with a price. The highest price that I paid to make my dreams come true was leaving my homeland and loved ones and staying far away for two years. So, I would like to dedicate this work to my beloved parents, Farangis and Farajollah, and my darling Atefeh, who had to undergo a one-year separation.

Chapter 1

Introduction

The rapid rise of technology, specifically the Internet, has led to a corresponding increase in the volume of online information, which has increased the demand to enhance users' information management capabilities. One of the significant developments in the information retrieval field is tied to the advent of recommendation systems, which aim to narrow down the domain of information for the users when exposed to a large amount of information. Recommendation systems have proven successful in various online sectors, including media (e.g., Youtube), sales (e.g., Amazon), and Recruitment (e.g., LinkedIn and Stepstones), by linking users to items of interest and fostering brand dedication.

Imagine, as an applicant, you would like to find a role that can perfectly match your profile (skills, experiences).Therefore, the relevant information and role must be quickly retrieved, though sifting through thousands of positions and roles to learn a small number of appropriate roles can be a tiresome endeavor for you. On the other hand, because there could be thousands of roles, as an employee or recruiter, it would be cumbersome for you to determine which role is the most relevant role for an applicant. Thus, a role recommendation system could be a practical system that allows for quick relevant information retrieval.

Given this objective, this thesis proposes, in partnership with Euranova, a role predictor and a top-k recommendation system that recommends the applicant's next possible roles based on his or her experiences and skills. This approach, unlike others, treats an applicant's experiences as a time-dependent list of roles and recommends a role for the applicant based on this time-dependent list. It is a truth that the applicants' experiences are developed through time; for instance, a candidate may begin his or her employment as a junior after graduation and subsequently attain seniority after several years of working. Consequently, this evolutionary trend is a fact that has been refuted by other works [5, 6, 7, 8].

As data is a crucial component of any learning model, the present work utilizes Curriculum Vita Data (CV Data) provided by Euranova to build the role predictor and top-k recommender. The CV Data contains 18937 applicants' CVs with 42216 available unique roles. Using the CV data, the domain of the proposed model will be IT-related.

The remainder of the thesis is structured as follows. In chapter 2, the background is provided for graphs, their applications, and machine learning on graphs. As this thesis focuses primarily on deep learning on graphs for role prediction, chapter 2 examines several deep learning algorithms for role prediction or link prediction in general. In chapter 4, the suggested model for role prediction is presented, and each component of its pipeline is

elaborated upon. In the final chapter 5, the conclusion and future work are presented.

Chapter 2

Background

Graphs can model many artificial or natural systems, phenomena, and interactions. In a graph, nodes (or vertices) represent distinct elements or items, and the connections between the elements or items are considered links (or edges). (In this thesis, we prefer to use the terms node and link). According to *Newman* [9, 10], the eminent Swiss mathematician *Leonhard Euler* conducted the first graph study in the 18^{th} century. He proposed the Königsberg Problem, which is regarded as the first genuine demonstration of graph theory. Since the twentieth century, graphs and their applications have been widely employed in several areas such as mathematics, physics, biology, economics, sociology, and computer science. Transportation Networks, Biological Networks, Social Networks, Food Webs, Metabolic Networks, and Blood Vessels are just a few of the many acceptable applications of graphs.

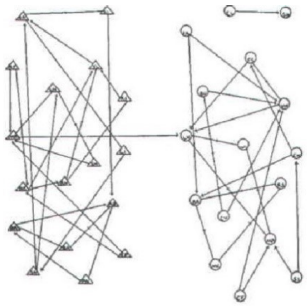

Figure 2.1: A graph representing the network of friendship among the students. On the right, girls (circles) and on the left, boys (triangles) [1]

Graphs are omnipresent and have been used in scientific works both explicitly and implicitly up to this point. For example, *Jacob Moreno*, a Romanian-American psychiatrist, used a graph structure to study the pattern of friendship among girl and boy students in a class in 1930s [1]. *Newman* claims that[10], while Moreno's work was incomplete and did not meet current standards, it might be regarded as the foundation of sociometry, which later became known as social network analysis. In 1951, *Myers and Shultz* did graph-based research on textile workers to determine how they sought work [11]. Similar

graph-based research on communities, acquaintance patterns, and friendship relationships has been conducted in [12, 13, 14, 15].

After the Internet became a universal public network in the early 1990s, the dependency of all human activities on the Internet on such a colossal scale and in such a short time signals the beginning of a new historical age in different disciplines of science, technology, and especially in the way humans contemplate and communicate. This vital mechanism now directly impacts many of our relationships and interactions. Communication expenses have fallen, and people's communication styles have altered considerably since e-mail services and the growth of online social networking websites like Twitter and Facebook. For example, people contribute their thoughts, attitudes, and perspectives on everyday concerns through these social services. Moreover, people no longer look for jobs using traditional methods; instead, they now use online job boards such as LinkedIn, Stepstone, and Job teaser to locate their dream jobs or be employed, as they have become an inevitable part of the modern recruitment market. As a result of these critical developments, researchers have been able to collect many graph data and employ algorithms and mathematics to analyze complex networks and even develop brilliant graph-based solutions.

2.1 What is a Graph?

Computers and algorithms are currently resolving numerous graph-related problems. Since the emergence and pervasiveness of graphs and complex networks, this necessity has consistently been recognized; therefore, computer scientists and researchers strive every year to build more effective problem-solving strategies. However, the question that pops up is, "How many varieties of graphs are there based on their structure?" Furthermore, "How is a graph saved on computers so that algorithms can be implemented?" Therefore, it is essential to provide a more rigorous definition of graph data and discuss various graph types. In the remainder of this section, different types of graphs that this thesis mostly revolves around are described. *Many of the materials in this section are adapted from* [9, 2, 16, 17].

2.1.1 Static Graph Definition

Mathematically, a static graph, denoted by the notation $G = (V, E)$, is generally characterized by a collection of nodes, referred to as V, and a collection of links referred to as E, which connect these nodes. For example, we indicate an outgoing edge from node $u \in V$ to node $v \in V$ as $l_{uv} \in E$. Furthermore, The k-hop neighborhood of a node $v \in V$, denoted by N_v^k, is the set of nodes at a distance less than or equal to k from v — for $k = 1$, $N_v^k = N_v$ is the set of node's neighbors.

Using an adjacency matrix $A \in \mathbb{R}^{|V| \times |V|}$ as a means of representing static graphs is a practical method. However, to accurately represent a graph using an adjacency matrix, we must first arrange the graph's nodes so that each node indexes a specific row and column in the adjacency matrix. The existence of links may therefore be represented as entries in this matrix, using the following syntax: $A[u, v] = 1$, if $l_{uv} \in E$, and $A[u, v] = 0$ otherwise. The adjacency matrix may contain additional information or distinct entries depending on the graph type for different purposes. Here, a few but more common types of static graphs are represented:

Simple Graphs

A simple static graph is a graph satisfying the three characteristics below.

- A maximum of one link should exist between each pair of nodes.
- There is no connection between a node and itself.
- A relation l_{ij} implies a relation l_{ji}.

Simple and undirected graphs model the majority of problems. It occurs when two nodes are either connected or disconnected. Numerous social and economic graphs, including partnership, friendship, and acquaintanceship, follow this pattern. 2.1 depicts a simple social graph.

Directed Graphs

The directed static graph is another sort of graph. The linkages between two nodes are asymmetric in directed graphs, meaning a relation from node i to node j does not necessarily mean a reciprocal relation from node j to node i. Take, for example, the World Wide Web. In this network, page x may have a hyperlink to page y, but page y may have no hyperlink to page x.

Weighted Graphs

Some static graphs may assign a positive or negative numerical value or weight to each link. For instance, on the Internet, the weight on link l_{ij} could reflect the amount of data flow or bandwidth between nodes i and j. In addition, a weighted link in a social network can represent the number of repeated calls between network members. Thus, if the adjacency matrix presents the weighted graph, the entries are arbitrary values ($A[i, j] \in R$), showing the weights of the links.

Graph with Features

In many real-world cases, the static graphs may have features or attributes at the graph, node, or link levels. For example, a graph modeling the connectivity in an online social network like Facebook, where the nodes are users and the links stand as the mutual relationship, may include the nodes' features, such as age, gender, or even users' postings. Moreover, in a citation network, where the nodes are the papers and the directed linkages show who cites whom, each node may have the paper's keywords as its features[18]. In case the graph benefits from features at the node/link level, a feature matrix $X \in \mathbb{R}^{|V| \times d}$ $/X \in \mathbb{R}^{|E| \times d}$, which usually corresponds to some representations (e.g., one-hot encoding for categorical variables), should accompany the adjacency matrix, where d is the feature dimension. In a **node feature matrix**, $X[i]$ corresponds to the feature vector of node i, and in a **link feature matrix**, $X[l_{ij}]$ denotes the feature vector of link l_{ij}.

2.1.2 Dynamic Graphs

A. Zaki et al. [16] define graph dynamicity from two perspectives: Topological Evolution and Attribute Evolution.

- **Topological Evolution:** The structural components of a dynamic graph, namely its nodes and links, alter continuously over time, thereby changing the graph topology continuously. Links can be added or removed, or a new node may be added while

others are removed. While link removal may not affect the existence of involving nodes, the node's removal causes all links associated with the node to disappear, as the existence of the links is conditioned on the node's presence.

- **Attribute Evolution:** It is associated with dynamic alterations to a graph's link or node features. The following actions represent these modifications: *add attribute value*, *remove attribute value*, and *update attribute value*.

Consequently, the dynamic graph may include both evolution classes or a mix that lies in the middle. In order to model graph evolution, two prominent models have become of great interest to researchers, specifically in the deep learning community: Discrete-time dynamic graphs (DTDG) and Continuous-time dynamic graphs (CTDG).

Discrete-time dynamic graphs (DTDG)

DTDG represents dynamic graphs by modeling them as a series of snapshots with the equation $G_{[t_1,t_n]} = \{G_1, G_2, G_3...., G_n\}$ [16, 19, 20]. Each snapshot, or G_i, is indeed a static graph that is used to represent the state of the dynamic graph at any time t_i. With a time point t_i, the snapshot is denoted by a triple $< V_i, E_i, t_i >$.

Two drawbacks are associated with DTDG:

- **Space Complexity**: DTDG modeling requires a huge amount of memory as each snapshot should be stored to capture the evolution. Each G_i may possess $O(|V_i|^2)$ links where V_i is the set of nodes in G_i; thus the space complexity of G_i would be $O(|V_i|^2)$ if a naive graph representation is used as described in 2.1.1 . If we assume $V_i = V$ for all $i \in n$, the total required space is bounded above by $O(n \times |V|^2)$, which is insufficient for large dynamic graphs modeled by many snapshots.
- **Approximation**: DTDG approximates the graph evolution as it may repudiate the dynamicity of the graphs, which holds vital temporal information regarding the graph evolution. For example, a link may disappear and then appear again later. Thus, a series of snapshots could miss the evolution as they may happen to only catch the link's presence or absence when taking the snapshots within different time windows.

Continuous-time dynamic graphs (CTDG)

CTDGs are a generic yet more complex framework to model dynamic graphs. In the literature, various CTDGs with different notations have been proposed [16, 19]. However, as this thesis principally deals with the TGN model proposed by Rossi et al. [2], CTDG proposed in their paper, called *temporal graph*, is presented here with a slightly different notation to maintain consistency across the document.

A temporal graph is a series of events $G = \{x(t_1), x(t_2), ...\}$ each of which could be a node/link topological or attribute evolution at a timestamp t_i. In the case of a node evolution event (**node-wise event**)[1] at time t denoted by $v_i(t)$, the event constructs a node with an index i and the specified features v if the node i does not already exist in the system; else, it updates the features v. A link evolution event (**Interaction Event**)[2] between nodes i and j is characterized by a (directed) temporal link $e_{ij}(t)$ where e denotes the associated link features— also called interaction event *message* in practice. With this setting, $V(T) = \{i : \exists v_i(t) \in G, t \in T\}$ is a temporal set of nodes, $E(T) = \{(i, j) : \exists e_{ij}(t) \in G, t \in T\}$ is a temporal set of links, and $N_v^k(t)$ is k-hop neighborhood for node

[1] In the original paper [2], the node evolution event is called **node-wise event**.

[2] In the original paper[2], the link evolution event is called **Interaction Event**.

$v \in V(T)$. Since temporal graph G is a more general form of dynamic graphs, a snapshot of the temporal graph at time t could be obtained as $G(T) = (V[0,t], E[0,t])$. *It is worth mentioning that this definition does not dictate the existence of only one link at any time. Thus the temporal graph could also be a multiple-temporal graph.*

2.1.3 Homogeneity vs. Heterogeneity

Not all graphs are homogenous. In numerous real-world situations, the graph may contain distinct groups of nodes, each of which represents a unique type of node with specific features and information associated with its nodes and links. The ogbn-mag dataset [21] is an example of a widely used heterogeneous benchmark graph consisting of four separate node types (paper nodes, author nodes, institution nodes, and field of study nodes) and four types of directed links connecting two types of entities. Moreover, one of the most common heterogeneous graphs is the Bipartite graph consisting of two different node types. Such heterogeneous graphs have been frequently employed in the development of both static [22] and dynamic [23, 24] graph-based recommendation systems.

2.2 Machine Learning on Graphs

Machine Learning is a data analysis approach that eliminates the need for human intervention in developing analytical models. More clearly, machine learning relies upon the concept that computers can automatically learn from data, determine patterns, and make judgments. Similar to Texts, Images, and Tabular Data, machine learning can be applied to graphs in supervised, semi-supervised, self-supervised, and unsupervised manners for various prediction tasks. However, unlike other data types, when the word machine learning is tied to graphs, its concepts would become slightly different. For example, in tabular data containing patient data, each personal information is considered independent of others. In contrast, the nodes — e.g., nodes as patients— in the graphs must be reliant and dependent on one another, as the graph has been generally proposed to reflect dependency among its elements.

Consequently, the graph's topology —and dynamicity if interested— should be taken into account when building a machine learning model on the graph. A standard approach is to compute an embedding z_i *(latent representation)* for each node in the graph, which can encode the node's temporal, topological, and associated features into a low-dimensional vector. This latent representation can then be used to solve many downstream tasks like node classification. Note that as the graph could undergo topological and attribute evolution in dynamic setting, the node embedding will thus change over time, denoted by $z_i(t)$.

In what follows, three main machine learning tasks on graphs will be briefly discussed. Further details on latent representation and its computation will be given in chapter 3.

2.2.1 Community Detection and Clustering

The study of the graphs modeling real-world networks has revealed the three most prominently observed properties that many real-world networks exhibit, namely small-world phenomena, scale-free, and high clustering coefficient [9]. The small world is the hypothesis that the average distance tends to be small in most real-world networks. The scale-free feature indicates that degrees of nodes in a network follow a right-skewed degree distribution with a fat tail. In other words, the network connectivity is dominated by relatively

few high-degree nodes. The high clustering coefficient explains that connected nodes have more joint neighbors in real-world networks than in networks where link formation appears to be independently random. Besides the properties mentioned above, a very recently discovered property called community structure, or so-called clustering, has been widely studied in the literature for both static and dynamic settings [25, 26]. Generally speaking, the community structure in a network may exist because the nodes have a strong tendency to be linked to other nodes having something in common. For instance, in society, the people who have similar attitudes, the same language, related jobs, or even who live in the same city probably would rather have closer relationships than with those who share nothing. Technically, although there has been no exact definition for the communities, a community is typically referred to as a set of nodes representing a dense connection between each other and a sparse connection to the rest of the network [27]. Extracting the communities disclose beneficial information regarding the structure of the networks depending upon their types. For example, (i) in a collaboration network, what scientists closely work on together [10], (ii) in a purchasing Network like Amazon.com [28], what items are more frequently purchased by the same buyers, (iii) how the terrorist cells can be formed and then identified [29], and so on.

2.2.2 Node Classification

In the graph, nodes can have labels that could represent the type or the class of the nodes. For example, presume a set of nodes in a graph are labeled, and the rest are still unlabeled. Then, node classification aims to predict the type of the unlabeled portion of the graph or, in some cases, predict the type of the unseen future nodes [2, 30, 31]. In the straightforward approach, if the embedding z_i for all nodes is obtained, node classification is achievable using a classifier (e.g., SVM, Logistic Regression, etc.) on the labeled nodes —deemed training nodes– to predict the type of the unlabeled nodes —deemed test nodes [32, 33]. In some circumstances, the node classification is coupled with node embedding and the optimization of the classifier is jointly performed — chapter 3 will cover this (joint) end-to-end learning for link prediction task.

2.2.3 Link Prediction

Like any phenomena in the universe, the graphs can evolve since they are generally utilized to describe the phenomena and their interactions (e.g., an online social network where users actively find new friends and form friendship interactions). The link prediction problem traditionally aims to extend the input graph in terms of links by anticipating the links that may arise in the future. In other words, depending on the graph's existing linkages and node features at time t or the history of graph evolution, we seek to maximize the probability of potential links between pairs of nodes.

Generally speaking, Link prediction is usually used for two graph-related tasks:

- **Analysis of graphs (networks) with incomplete data:** Link prediction helps identify potential hidden links in a network or detects spurious links. For example, a limited number of linkages in biological networks, such as protein-protein networks, may be retrieved [reference]. Suppose the link prediction on these networks is appropriately accomplished. In that case, the most likely connection can be discovered, possibly removing the need to check the actual links.

- **Link prediction as a recommendation system:** Recommendation systems are one of the link prediction applications that could be a driving force in enhancing client happiness. Although social networking websites and online retailers can provide users with a wide range of information and systems, it is essential to note that this enormous amount of data may cause consumers to become confused. Therefore, recommendation systems have been proposed to recommend appropriate products, services, and other items to users based on their profiles and item information. As a result, many internet networking systems now include recommendation systems, which are frequently used for advertising, commercial applications, and employment-oriented online services, among other things. In a straightforward case, suppose a bipartite graph that models a user-item relationship where there exists a link between a user-item pair if the user has shown his/her interest in the item (e.g., a product, a movie, etc.). Thus, in such a circumstance, link prediction between users and items can be deemed the heart of a recommendation system [22, 23].

  The link prediction based on static graphs has proven inefficient [26, 34] since it rejects the evolutionary pattern of the graphs. For example, in a bipartite applicant-role graph, an applicant may take a role x at a time t, then go on sabbatical for a year, and re-take the role x after. Thus, a link predictor based on only a snapshot [35], would not be able to catch this pattern. Assuming the embedding z_i for all nodes is at hand — $z_i(t)$ for temporal graph—, a link prediction between two nodes i and j would be possible by computing a similarity function like *Cosine Similarity* or a *decoder —if end-2-end learning is employed, typically an MLP decoder that receives the embeddings of two nodes of interest* [2, 34].

2.2.4 Transductive Learning vs. Inductive Learning

In Machine Learning, *Transductive Learning* is a learning paradigm in which both the training and testing datasets are seen during training, but the testing dataset is not used to calculate the loss. In contrast, in *Inductive Learning*, only the training data are observed during model training, and the trained model is then used to perform a specified task (e.g., classification) on unseen test data.

When applying an algorithm for machine learning to graphs, it is important to compute an embedding for nodes so that the algorithm can use the embedding for any prediction task. Since, in some cases, the embedding technique must capture the entire topology of the graph in order to compute embeddings, it may only allow transductive learning. A training node may be adjacent to a test node, and it may be necessary to collect data from the training node's neighbors, including test nodes, in order to compute the embedding for the training node. In contrast, the test nodes should learn about their neighbors, which could be training nodes to generate the embedding for the test nodes. As a result, if the embedding technique is strongly dependent on the network topology, it may only give transductive learning because the embedding procedure would require the entire graph topology in order to provide embeddings for all nodes, whether training or test.

Nevertheless, an embedding technique may not be dependent on the graph's topology, so that inductive test nodes are also possible. Neither the loss computation nor the embedding procedures are performed on these inductive test nodes during training, indicating that these nodes and all of their links are completely unseen. In inductive learning, the goal is to compute the unseen nodes' embeddings for any downstream task after the model is trained on the training nodes.

2.3 Natural Language Processing (NLP) for Graphs

The machine can only interpret numbers. However, in many practical contexts, nodes and links may contain text [21]. For instance, in the citation network, the paper nodes may contain abstracts as their text features. In addition, the nodes of a co-purchasing network contain the product description. Therefore, these textual characteristics should be transformed into machine-understandable features.

Utilizing available NLP embedding algorithms to produce a low-dimensional real-valued vector for text is a common practice. For example, word2vec [36], Glove [37], and Bert [38] are widely applicable embedding algorithms that compute an embedding for every word in the corpus. Thus, a second procedure is required to compute the vector for the entire text (e.g., averaging all the word embedding)

Doc2vec [39] and Sbert [4], unlike the previously stated embedding approaches, enable direct embedding for the entire text. Sbert is an elegant and cutting-edge text embedding that enables the computation of text embeddings for over 100 languages.

Notably, many pre-trained models exist for all embedding strategies, enabling them to be trustworthy, meaningful, and low-dimensional. By adding embedded text data, node features enhance the efficacy of graph embedding. Numerous graphs that depict the real world include this kind of data.

Chapter 3

Deep Learning for Graph Link Prediction

As explained in Section 2.2, to train a machine learning algorithm on static or dynamic graphs for graph-related tasks, for each node i, a low-dimensional real-valued vector z_i, called node embedding, must be calculated to preserve the node's structural and (temporal) information as well as its content features.

Many node embedding approaches have been proposed— *an interested reader may refer to [40] for static graphs and [41] for dynamic graphs*—, allowing us to achieve the objective above. Node2Vec [33], for instance, is a popular node embedding approach *for static graphs* that maps nodes' positions in the graph into an n-dimensional latent space such that the probability of preserving the network neighborhood of nodes in the latent space is maximized. This node embedding facilitates the use of off-the-peg machine learning algorithms for many graph machine learning tasks, such as node classification — typically, a similarity measure such as *Cosine Similarity*, *Euclidean Distance*, *Manhattan Distance*, etc., is used to calculate the distance between nodes for link prediction.

Nonetheless, in this case, the pipeline comprises at least two distinct phases: (i) node embedding computation and (ii) training by a machine learning algorithm. Thus, the performance of the machine learning algorithm does not influence the node embedding, whereas the embedding does affect the machine learning performance, thereby increasing the complexity of the pipeline due to the fact that the embedding can be acquired and processed differently.

Deep learning has demonstrated unprecedented performance in numerous research disciplines, including Natural Language Processing, Computer Vision, and Signal Processing. Graphs are not exceptional and have attracted significant attention from the deep learning field under *graph neural network (GNN) formalism* in order to address graph-related challenges. GNN is an end-to-end graph learning system that permits the training of complicated graph problems by applying gradient-based learning to the entire system.

In this chapter, end-to-end learning and a few common GNNs for both static and continuous-time dynamic graphs (CTDGs) are introduced. These GNNs are — *to the best of my knowledge although there are many*— the most prevalent models in the field and represent important milestones. In addition, understanding them aids the reader in digesting the information presented in subsequent chapters — *if they are unfamiliar with the topic*. Lastly, although these GNNs have many applications and can be used for

different graph tasks, this chapter is only confined to the link prediction task due to the nature of this thesis.

3.1 End-to-end Learning for Link Prediction

Generally speaking, an end-to-end learning system for link prediction consists of self-supervised learning with an encoder-decoder framework [42, 43]. In the encoder-decoder framework, the learning problem is viewed as requiring two essential processes: an encoder that maps graph nodes into low-dimensional embeddings and a *(pairwise)* decoder that utilizes the embeddings to rebuild the neighborhood of the nodes as observed in the input graph.

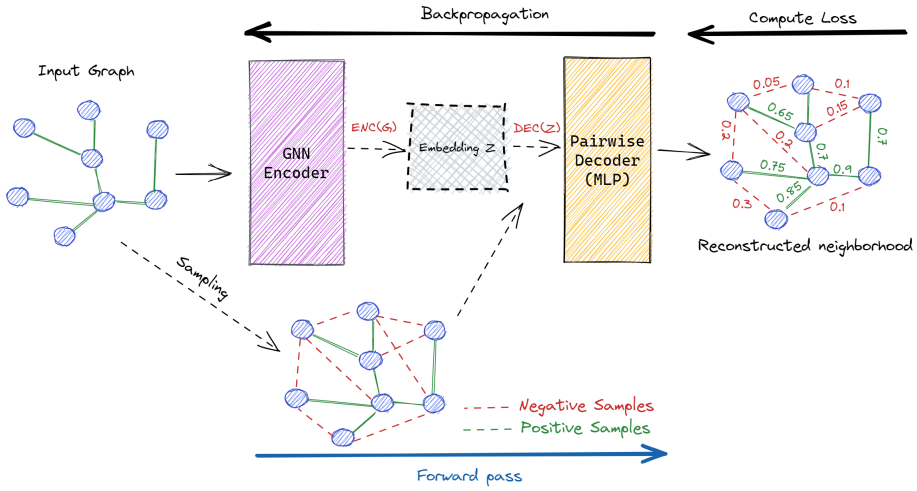

Figure 3.1: An end-to-end encoder-decoder framework for link prediction with negative sampling. Numbers on the reconstructed neighborhood are the link probabilities.

In end-to-end learning for link prediction, GNN encoders with varied architectures and techniques are utilized depending on the type of the input network (e.g., static or dynamic graphs). In contrast, the decoders typically adhere to a simple neural network architecture with minor variations. Consequently, a typical decoder for the link prediction task will be detailed below. In addition, various encoders will be discussed in the respective sections 3.2 and 3.3.

3.1.1 Decoder

Suppose we have a GNN encoder ENC that can, regardless of the GNN model used in the encoder, receive the gradient during training, as represented in Fig. 3.3, and map the nodes into the latent space $Z \in R^{|V| \times d}$. A pairwise decoder DEC then attempts to rebuild the node neighborhood by using two embeddings, z_i and z_j, as inputs —a concatenation of z_i and z_j— and predicting the node neighborhood. Prediction in the decoder can be

made using MLP with only one unit in the output layer (with sigmoid activation)— several hidden layers, dropout layers, and regularization are possible. The objective of this setting is to increase the probability of existing links and reduce the probability of nonexistent links. Notably, given that only existing linkages are available, negative sampling, which samples a subset of node pairs as negative pairs, is required. The decoder must then differentiate between the existing and nonexistent links when taking the two embeddings as input. Accordingly, the link prediction problem on the decoder side resembles a binary classification task; hence, any loss function used for the binary classification (e.g., binary cross-entropy loss) can be utilized for optimization as follows [17, 43]:

$$L = \sum_{(i,j) \in E_{train}} loss(DEC(ENC(i), ENC(j)), A[i,j]) \tag{3.1}$$

where E_{train} is the training set that includes positive and negative samples. Moreover, if the input graph is dynamic and includes the temporal links, Eq. 3.1 is turned into Eq. 3.2:

$$L = \sum_{(i,j,t) \in E_{train-temp}} loss(DEC(ENC(i,t), ENC(j,t)), A[i,j,t]) \tag{3.2}$$

where $E_{train-temp}$ is a set of negative and positive temporal links for training, $ENC(i,t)$ computes the embedding $z_i(t)$ for node i at time t, and $A[i,j,t]$ is equal to 1 if there exists a (directed) link between i and j at time t, otherwise 0.

3.1.2 Training

In order to train the model in an end-to-end fashion, encoder and decoder parameters are initialized at random. The encoder computes the node embeddings at each epoch, and the pairwise decoder predicts the linkages based on the input embeddings. In the interim, the loss function is computed, and the model parameters are updated once receiving the gradients computed by an optimization algorithm (e.g., Adam [44]). This process is continued until a certain condition is met (e.g., a fixed number of epochs). This end-to-end learning system is similarly applied to the dynamic GNN models— *though the embedding $z_i(t)$ at any time t is computed using a dynamic encoder like TGN [2].*

3.2 GNN for Static Graphs

GNN has its root in the neural message passing framework (or message passing) [17]. The message passing is established on the concept of the convolution process used for grid data (e.g., images) and aims to generalize it to graph data. In the message passing, the objective is to exchange the nodes' information among the nodes in the local neighborhood so that the nodes' embedding includes the nodes' structural information as well as information about all the features in the nodes' k-hop neighborhood.

Formally, given that $h_i^{(k)}$ is the hidden embedding of the node i such that $h_i^{(0)}$ is equal to the feature vector of i, at each iteration $k > 0$ of message passing, $h_i^{(k)}$ gets updated as follows:

$$h_i^{(k+1)} = F(h_i^{(k)}, AGGREGATE^{(K)}(h_j^{(k)}, \forall j \in N_i)) \tag{3.3}$$

where F and $AGGREGATE^{(K)}$ are differentiable functions. $AGGREGATE^{(K)}$ is a permutation-invariant set function (e.g., Summation or Averaging function) that integrates the hidden embedding of neighboring nodes into a vector. In addition, function F merges

the aggregated embedding obtained by $AGGREGATE^{(K)}$ with the hidden embedding of the node i to compute $h_i^{(k+1)}$. In some cases, the message passing includes the self-loop; thus the hidden embedding is computed by:

$$h_i^{(k+1)} = F(h_i^{(k)}, AGGREGATE^{(K)}(h_j^{(k)}, \{\forall j \in N_i \cup i\})) \tag{3.4}$$

Throughout each iteration of message passing, a hidden embedding is produced for each node i, as shown in Eq. 3.3 or 3.4. Each iteration k is considered a message passing layer k. After the computation of multiple layers, let's say K layers, the output of the final layer K is used as the node embedding Z. Consequently, each embedding $z_i \in Z$ contains not only the structural information for node i but also information about all the features in the node K-hop neighborhood.

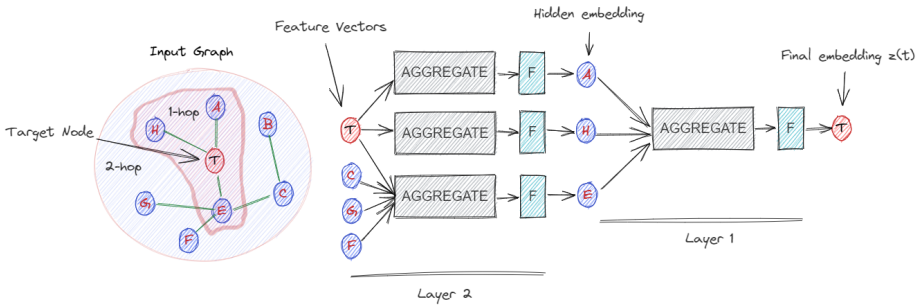

Figure 3.2: Message Passing framework with two layers. The image illustrates the computation of embedding z_t for a target node t using message passing with two layers. In the input graph, the blue ball represents the 2-hop neighborhood, and the red region shows the 1-hop neighborhood (direct neighbors).

Even though this message passing framework only provides an abstract view of the static GNNs, it is essential to note that various GNNs based on it have been proposed, the explanation of which is beyond the scope of this thesis. The interested reader is encouraged to read [17] and [45] to thoroughly understand this framework and its variations. However, the remainder of this section discusses a few must-learn static GNNs that are somewhat adapted from [17].

3.2.1 First GNN

Scarselli et al. [46] introduced the fundamental GNN model, which is in a number of respects analogous to the standard message passing framework. In this GNN model, the information in the local neighborhood of a node i is aggregated by adding all the embeddings of node i's neighbors, which were obtained at level k. The aggregate outcome is then mixed with the current embedding of the node i. As with the traditional MLP, a non-linear function such as $ReLU$, $Tanh$, or $Sigmoid$ is employed to produce the hidden embedding for node i at the level $k+1$. The basic GNN is formulated as:

$$h_i^{(k+1)} = \sigma\left(W_{self}^k h_i^{(k)} + W_{neigh}^k \sum_{j \in N_i} h_j^{(k)} + b^{(k)}\right) \tag{3.5}$$

where W^k_{self}, $W^k_{neigh} \in \mathbb{R}^{d^{(k)} \times d^{(k-1)}}$ are learnable parameters, $b^{(k)} \in \mathbb{R}^{d^{(k)}}$ is the bias term, and σ is a non-linear function. Although in Eq.3.5,W^k_{self}, W^k_{neigh}, and $b^{(k)}$ are assumed to be trained independently at each level k, these parameters can be shared and trained across all levels.

According to Equation 3.5, the computation of the hidden embeddings is highly dependent on the graph's topology because the features of all neighbors are aggregated by a summation. To build the embedding for a target node i, the model must observe all its adjacent nodes regardless of whether they are training or test nodes. Overall, it implies that the basic GNN lacks *inductive learning* and provides only *transductive learning* for any graph prediction task (e.g., link prediction).

3.2.2 Graph Convolutional Network

Convolutional Graph Network (GCN) proposed in [47] is a building blocks of GNNs. This networks take advantage of the message passing framework with self-loop and the normalized aggregation function. The GCN is formulated as:

$$h_i^{(k+1)} = \sigma\left(W^k \sum_{j \in N_i \cup \{i\}} \frac{h_j^{(k)}}{\sqrt[2]{|N_i||N_j|}}\right) \tag{3.6}$$

where W^k holds learnable parameters and σ is a non-linear function such as $ReLU$ or $Tanh$.

GCN is a topology-dependent model incapable of inductive learning, similar to the basic GNN in Eq. 3.5. On the basis of the GCN, inductive learning models such as Graph **SA**mple and aggre**G**at**E (GraphSAGE)** have been proposed in [30]. GraphSAGE is trainable on a limited subset of the graph and can be generalized to unseen nodes. GraphSAGE is based on the concept of sampling a set neighborhood of a node to learn the node's hidden embedding, as opposed to working over all k-hop neighborhoods in GCN. Different aggregating functions, including the mean aggregator, LSTM-based aggregator, and maximum pooling aggregator, are proposed in the original study. In GraphSAGE, the learnable weights are not node-specific; therefore, they are shared across all neighborhoods. GraphSAGE establishes its concept on the fact that graph representations could contain structures that enable us to make predictions about a node based on its surroundings. This allows GraphSAGE to compute the embedding for an unseen node when it appears by applying the shared weights to the sample neighborhood of the unseen node.

3.2.3 Graph Attention Network

The attention mechanism has indisputably been one of the most remarkable advances in deep learning. It is a technique that has given Natural Language Processing (NLP) and Image processing a distinctive climacteric. Current state-of-the-art solutions for NLP tasks, such as Translation, are noted to be primarily based on the attention mechanism. For instance, in the Translation problem, the attention mechanism enables the model to learn how much attention it should give to the words in a source sentence while translating a particular source word and predicting an equivalent target-language word. Similarly, in Graph Attention Network (GAT) proposed by Veličković et al. [31], the fundamental concept is to provide an attention weight to each neighbor, which is used to measure the neighbor's impact during the aggregation process.

The authors proposed the attention α_{ij} that indicates the importance of node j's features to node i as follow:

$$\alpha_{ij} = \frac{exp\left(LeakyReLU\left(\vec{a}^T[Wh_i^{(k)}||Wh_j^{(k)}]\right)\right)}{\sum_{j\in N_i} exp\left(LeakyReLU\left(\vec{a}^T[Wh_i^{(k)}||Wh_j^{(k)}]\right)\right)} \tag{3.7}$$

where $.^T$ denotes transposition, $||$ represents the concatenation operation, $\vec{a} \in \mathbb{R}^{2d^{(k+1)}}$ is a weight vector, and $W \in \mathbb{R}^{d^{(k+1)}\times d^{(k)}}$ is a weight matrix for a shared linear transformation that is applied to every node.

After computing the normalized attention coefficients, GAT uses the coefficients to define the next level's hidden embedding as a weighted sum:

$$h_i^{(k+1)} = \sigma\left(\sum_{j\in N_i} \alpha_{ij} W h_j^{(k)}\right) \tag{3.8}$$

where σ is a non-linear function.

GAT supports multi-head attention, as do all attention mechanisms utilized for NLP and Images. Although the general approach suggests concatenating the results of heads for obtaining the hidden embedding, the authors recommend averaging the results, specifically, when performing multi-head attention on the final layer of the network. Thus, the hidden embedding is acquired as follows:

$$h_i^{(k+1)} = \sigma\left(\frac{1}{p}\sum_{p=1}^{P}\sum_{j\in N_i} \alpha_{ij} W^p h_j^{(k)}\right) \tag{3.9}$$

where P is the number of the heads.

In GAT, the attention mechanism is applied in a shared fashion to all links of the graph; consequently, unlike the basic GNN and GCN, it is not a topology-dependent model, allowing GAT to be used for inductive learning.

3.3 GNN for CTDGs: Temporal Graph Network

Temporal graph network (TGN), devised by E. Rossi et al. [2], is a generic inductive framework of dynamic Graph that can learn the temporal embedding on continuous-time dynamic graphs, aka CTDGs. Thanks to an RNN[1]-based memory module, TGN can memorize the node's history as a compressed state for each node i, which is then utilized to generate temporal embedding of the node $z_i(t)$. Accordingly, based on the computed temporal embeddings $Z(t) = \{z_1(t), z_2(t), ..., z_{|V(t)|}(t)\}$, any possible downstream tasks on CTDGs (e.g., Link Prediction as interested in this thesis) is achievable.

TGN comprises multiple steps to fulfill the stated objective. For each node i, a memory, aka the node's state denoted by $s_i(t)$, is used to summarize the history of the node up to time t. If a node i is observed for the first time during a link evolution event (*interaction event*) in which a temporal link $e_{ij}(t)$ is added to the graph, the memory of the node i is initialized with a zero vector; otherwise, it gets updated by a memory updater as explained below. Additionally, because the memory for each node is simply a non-learnable vector, it is possible for it to be updated during test time when the model is fed new interactions — *even when the model has not seen the node during training, its memory can still get updated at test time because the memory is first initialized as a zero vector once the node appears.*

[1] Recurrent Neural Network

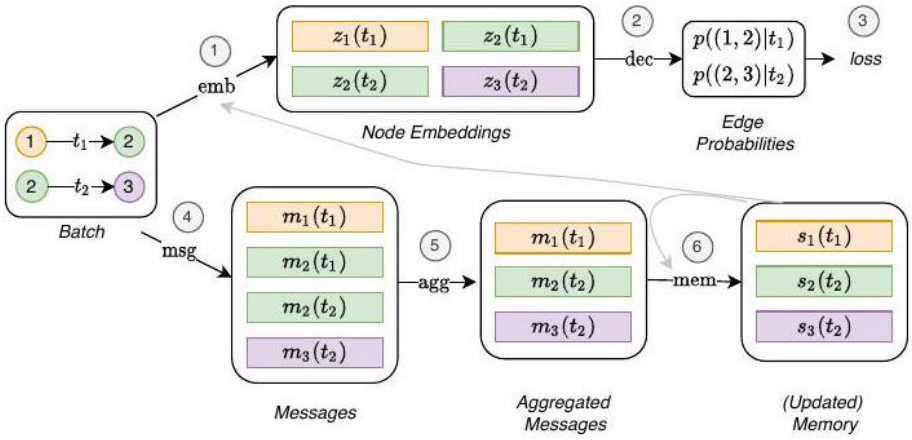

Figure 3.3: Computations performed by TGN on a batch of time-stamped interactions. Adapted from [2].

For updating the memory, two messages are computed for the nodes participating in the interaction event as follows:

$$^{2}m_i(t) = (s_i(t^-)||s_j(t^-)||\Delta t||e_{ij}(t)), \quad m_j(t) = (s_j(t^-)||s_i(t^-)||\Delta t||e_{ij}(t)) \tag{3.10}$$

where t^- denotes the time just before time t,|| is the concatenation operation, and Δt is the time since the last interaction.

When training a large input CTDG, TGN can benefit from batch processing to improve the model performance in terms of memory and runtime. However, a node i may be observed in several events within a batch, resulting in several computed messages for node i. Thus, TGN utilizes an aggregation mechanism to only obtain one message for each node i in the batch. In the original paper, the aggregation mechanism is suggested to be implemented as either the most recent message— only picks the last message in the batch— or the average of the messages. With the aggregated message $\overline{m}_i(t)$, the memory of the node i is updated by

$$s_i(t) = mem(\overline{m}_i(t^-), s_i(t^-)) \tag{3.11}$$

where mem[3] is an RNN-based memory function such as **GRU** or **LSTM**. Since in an interaction event, two end-points i and j are involved, the memory should get updated for both nodes.

To generate temporal embeddings, the authors established a general form of embedding function upon which multiple learnable functions can be defined and implemented.

[2]In the original paper, the authors stated that the messages can be computed using more sophisticated learnable functions such as Multi-layer perceptrons. However, the concatenation operation was suggested as it simplifies the implementation.

[3]The authors used GRU in their experiments.

$$z_i(t) = \sum_{j \in N_i^k([0,t])} h(s_i(t), s_j(t), e_{ij}(t), v_i, v_j) \tag{3.12}$$

This general embedding module in Eq. 3.12 is presented as a solution to the problem of staleness in dynamic embedding models. The staleness issue arises because the node may remain dormant for an extended period. Therefore, the embedding should be obtained so that it has the most recent information about the node. The suggested embedding module computes the temporal embedding of the node by aggregating the node's temporal neighbors. Thus, even if the node has been idle for a long time, some of its active neighbors, along with their memories, may assist in keeping the node's embedding up-to-date.

According to the TGN paper's ablation study, the best embedding module is *Temporal Graph Attention* (**attn**), which obeys the general embedding function in 3.12. **attn** is a series of L-graph attention layers, which can learn which neighbors are the most influential based on their memory, features, and interaction time. Mathematically, **attn** is defined as:

$$\begin{aligned}
\mathbf{h}_i^{(l)}(t) &= \mathrm{MLP}^{(l)}\left(\mathbf{h}_i^{(l-1)}(t) \| \tilde{\mathbf{h}}_i^{(l)}(t)\right) \\
\tilde{\mathbf{h}}_i^{(l)}(t) &= \mathrm{MultiHeadAttention}^{(l)}\left(\mathbf{q}^{(l)}(t), \mathbf{K}^{(l)}(t), \mathbf{V}^{(l)}(t)\right), \\
\mathbf{q}^{(l)}(t) &= \mathbf{h}_i^{(l-1)}(t) \| \phi(0), \\
\mathbf{K}^{(l)}(t) &= \mathbf{V}^{(l)}(t) = \mathbf{C}^{(l)}(t), \\
\mathbf{C}^{(l)}(t) &= \left[\mathbf{h}_1^{(l-1)}(t) \| \mathbf{e}_{i1}(t_1) \| \phi(t-t_1), \ldots, \mathbf{h}_N^{(l-1)}(t) \| \mathbf{e}_{iN}(t_N) \| \phi(t-t_N)\right]
\end{aligned} \tag{3.13}$$

where $\phi(\cdot)$ represents a generic time encoding, $||$ is the concatenation operation, $\mathbf{z}_i(t) = \mathbf{h}_i^{(L)}(t)$, and $\mathbf{q}^{(\mathbf{l})}(t)$, $\mathbf{K}^{(\mathbf{l})}(t)$, and $\mathbf{V}^{(\mathbf{l})}(t)$ are learnable-parameters of MultiHeadAttention. The input to the l-th layer is the node i's hidden embedding $\mathbf{h}_i^{(l-1)}(t)$, node i's neighborhood hidden embeddings $\left\{\mathbf{h}_1^{(l-1)}(t), \ldots, \mathbf{h}_N^{(l-1)}(t)\right\}$, and the input representation of each node as $\mathbf{h}_j^{(0)}(t) = \mathbf{s}_j(t) + \mathbf{v}_j(t)$.

In the TGN paper, the following embedding modules are also presented, which can be used instead of **attn**.

- **Identity (id):** The embedding is equal to the memory of the node, meaning $z_i(t) = s_i(t)$.The identity function is plagued by the staleness issue. Because the memory of a node is only updated when the node is participating in an interaction, a long time of inactivity leads the memory of the node to become outdated, rendering the concept impractical in reality.
- **Time Projection (time):** The embedding is computed by a time-based function presented in [23], meaning $z_i(t) = (1 + \Delta t \mathbf{w}) \circ \mathbf{s}_i(t)$, where $\mathbf{w}$ are learnable parameters, Δt is the time since the last interaction, and o denotes element-wise vector product.
- **Temporal Graph Sum (sum):** A simpler and faster aggregation over the graph:

$$\begin{aligned}
\mathbf{h}_i^{(l)}(t) &= \mathbf{W}_2^{(l)}\left(\mathbf{h}_i^{(l-1)}(t) \| \tilde{\mathbf{h}}_i^{(l)}(t)\right), \\
\tilde{\mathbf{h}}_i^{(l)}(t) &= \mathrm{ReLu}\left(\sum_{j \in n_i([0,t])} \mathbf{W}_1^{(l)}\left(\mathbf{h}_j^{(l-1)}(t) \| \mathbf{e}_{ij} \| \phi(t-t_j)\right)\right).
\end{aligned} \tag{3.14}$$

  where $\phi(\cdot)$ is again a time encoding and $\mathbf{z}_i(t) = \mathbf{h}_i^{(L)}(t)$.

3.3.1 TGN as a generic framework

TGN provides a generic framework for training various popular dynamic models, including ones proposed before TGN. The JODIE algorithm [23], one of the most well-known dynamic models, was proposed in 2019, a year before TGN. JODIE's memory module is RNN-based, and embedding is performed using the Time Projection approach, as described in 3.3. In addition, another prominent algorithm, Dyrep [48] published in the same year, employs an RNN as its memory updater, and the memory is utilized directly for embedding. In addition, it adds a summary of the node's neighborhood to the message function using graph attention. Lastly, TGAT [34] is a special case of TGN in which memory and its associated modules are absent, and graph attention is utilized as the Embedding module—though in TGAT, for the first layer $\mathbf{h}_j^{(0)}(t)$, the inputs are just raw node features. Altogether, all these models can be implemented using the TGN framework with less effort (small modification required).

Since the advent of TGN, various approaches have been offered in an effort to enhance certain aspects of TGN. APAN [3], for instance, was developed to alleviate the latency issue of TGN's online inference. In addition, to increase the accuracy of link prediction using TGN, TGN with Causal Anonymous Walk was introduced in [49], which is slightly superior to TGN. Moreover, Adaptive Data Augmentation was proposed in [50] to alleviate the overfitting issue that may arise in some cases owing to the complexity of the TGN. Although the TGN enhancements mentioned above have not been used in this thesis, they are briefly described — without details— in the appendix A as they may prove useful in future work —in some respects— to improve the role prediction based on the CV data provided by our industrial partner, **Euranova**.

Chapter 4

Role Prediction

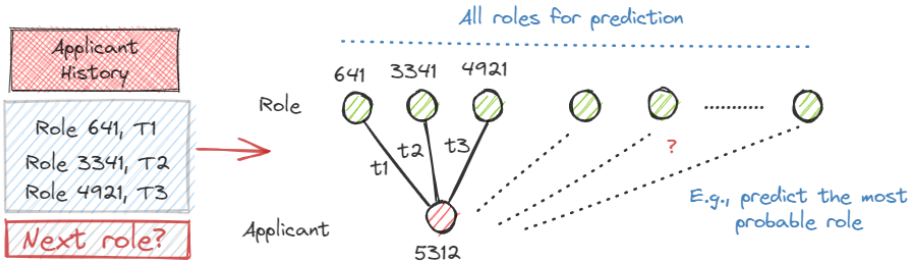

Figure 4.1: An example of role prediction task in the dynamic setting.

This thesis aims to propose a role predictor that can predict an applicant's subsequent role given a history of his/her experiences and skills. The phrase "History" incorporates "Time" expressly. Because the role prediction task generally attempts to predict the next roles for humans, and humans, their capabilities, interests, and skills are consistently undergoing changes over time, proposing a role predictor model that can capture the evolutionary pattern over time is, therefore, a necessity.

As discussed in section 1.1 and Chapter 2, one of the machine learning applications on the graphs is link prediction, based on which recommendation systems can be devised. Given the recent growth of Graph Neural Networks in terms of dynamicity and temporality, this thesis has attempted to utilize dynamic GNN models such as TGN to achieve the stated objective.

Because any machine learning models include many complexities (for example, many data preprocessing techniques) and much ambiguity due to lots of choice in the hyper parameters, this chapter proposes a pipeline, as shown in Fig 4.2, for the role prediction task given the CV data[1] provided by **Euranova** in the dynamic setting in the hopes of mitigating the model complexities and narrowing down the search space for the hyper parameters.

[1] Curriculum Vitae

The processing phase, in which the provided CV data is preprocessed in the appropriate manner, is the first phase in the pipeline. The second phase of the pipeline generates several different dynamic graph representations of the CV data. In the third phase, a number of different popular dynamic GNN models are trained in an effort to identify the most suitable possible graph representation of the CV data. In the fourth phase, the best model, which can lead to a better role predictor, is found through an ablation study, and a role prediction model is proposed. In the fifth and final step of the pipeline, the proposed model as a top-k role recommender is investigated as it appears to be more applicable to real-world settings. In the following sections of this chapter, we will go through each pipeline phase in greater detail.

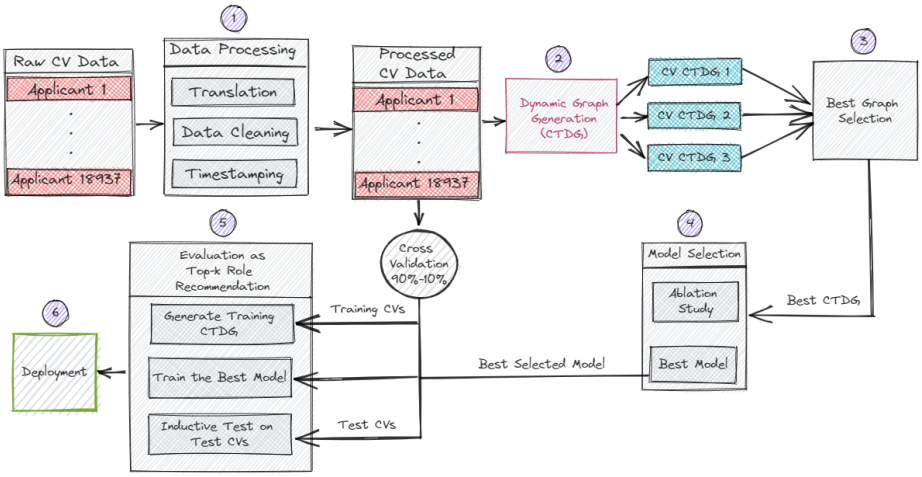

Figure 4.2: Role predictor pipeline.

Note: In this chapter, the term dynamic graph refers to as a continuous time-dynamic graphs, discussed in Section 2.1.2.

4.1 Raw CV Data

The CV Data include thousands of applicants' CVs, which provide pertinent information about the candidates. Each CV, as shown in Fig. 4.3, includes the Language the individual speaks, a list of prior experiences —*each including the role and the period the individual performed the role*—, top skills, publications, businesses that encodes the companies the individual has worked for, and experience-related skills **(skills-exp)**. Throughout the pipeline, the only information that is used is the candidates' experiences, language, Top-skills, and skill-exp. This is due to the fact that there is insufficient information offered regarding the business and publication. Table 4.1 shows the statistical summary of the CV data.

```
{'language': 'french',
 'experience': [{'role': 'senior java consultant',
   'period': 'june 2016-present'},
  {'role': 'senior master data developer (ebx mdm)',
   'period': 'december 2014-jun 2016 (1 year 7 months)'},
  {'role': 'developer java ee senior',
   'period': 'february 2014-november 2014 (10 months)'},
  {'role': 'developer java ee senior / project manager',
   'period': 'august 2011-january 2014 (2 years 6 months)'},
  {'role': 'java/j2ee development engineer',
   'period': 'may 2009-october 2010 (1 year 6 months)'},
  {'role': 'developer',
   'period': 'october 2005-march 2009 (3 years 6 months)'}],
 'topskills': ['java enterprise edition', 'spring', 'uml'],
 'publications': [],
 'business': ['35983', '10968', '5501', '13476', '0', '28484', '30755'],
 'skills_exp': ['java', 'linux', 'javascript', 'c', 'uml', 'mysql', 'oracle']}
```


Figure 4.3: A example of an available applicant's CV.

The number of available CVs	18937
The number of unique job roles	42216
The average number of job roles per each applicant	5.69
Ratio of the unique job roles to the whole job roles	0.39
The number of unique skills-exp	934
The average number of skills-exp per each applicant	17.69
The number of unique top skills	5900
The average number of top skills per each applicant	2.9
The average number of the language (mother tongue) per each applicant	1

Table 4.1: Statical Summary of the CV Data

4.2 Data Processing

As the initial phase of our machine learning pipeline, the data processing phase involves transforming unstructured input data into usable model data. Therefore, Euranova's CV data must be cleansed, preprocessed, and adequately prepared to aid in the role prediction task. For instance, in the CV Data, many role titles and job experiences were provided in languages other than English, posing a challenge for role prediction since many existing NLP algorithms have been developed for English. Moreover, although the time dimension is of considerable relevance to us in this research, the periods of the experiences in the CV Data do not adhere to the same structure — and the same language — making it difficult to extract the dates (timestamps) from the applicants' CVs.

This section will concern the information regarding the preprocessing of the CV data (e.g., role translation and data extraction).

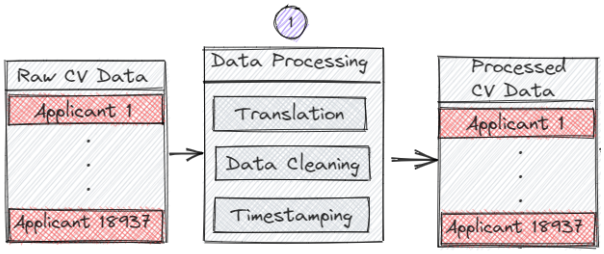

Figure 4.4: Data Processing Phase.

4.2.1 Translation

As the CV data were collected from candidates worldwide, notably from French-speaking districts, there is a significant likelihood of encountering applicant information in a language other than English. For instance, French is the most common language, and many input dates and candidates' experiences (*past roles*) are provided in French— around 60% of the applicants—, making date extraction and feature embedding difficult. Since most existing NLP techniques used for embedding and date extraction —*especially the pre-trained models*— are primarily compatible with English, the initial preprocessing step is to translate all the information to English.

The IBM Watson translation API[2] was initially utilized for translation. Nevertheless, in addition to the IBM Watson translation API, Microsoft Azure Cognitive Service[3] was used to assure the accurate language conversion, as numerous roles and periods were first mistranslated. Furthermore, due to the dynamic nature of this work, the translation of the periods (dates) was also double-checked to ensure that they were all translated accurately. To this end, all roles whose periods were not translated correctly or did not stick to the standard format were filtered out using a check procedure.

4.2.2 Data Cleaning

As with our CV data, the roles, periods, and skills (top-skills and skills-exp), which are text data, may contain punctuation and non-letter characters such as "/". As a general practice in NLP, all the words were lowercased to help preserve the flow of consistency during the NLP tasks. In addition, the majority of punctuation was removed for standardization, as punctuation can cause noise and uncertainty while embedding the texts.

Fig. 4.5 illustrates a word cloud being a visual representation of roles. By a cursory look at the Figure, it could be observed that most CVs are related to the IT industry, which may be because **Euranova**— *an IT service provider specializing in artificial intelligence, data-centric architectures, and product development*— collected the CVs from the candidates. Nevertheless, Fig. 4.5 reveals the domain of the roles based on which a role predictor is offered in this thesis.

[2]https://cloud.ibm.com/apidocs/language-translator
[3]https://docs.microsoft.com/en-us/azure/cognitive-services/translator/

Figure 4.5: A visual representation of roles highlighting frequent words and roles

4.2.3 Date Extraction and Timestamping

For each role that a person inputs as part of her/his work experience, there is a period indicating the length of time the individual held the position. Since the periods are expressed in human-readable formats and include human-readable dates *(e.g., June 2017-July 2018 or June 25, 2020-Present)*, the dates must be extracted and then translated to a machine-readable format compatible with the dynamic GNN models. In order to accomplish this, *the LexNLP date extraction* method proposed in [51] was employed, which provides machine learning-enhanced regular expression for date extraction.

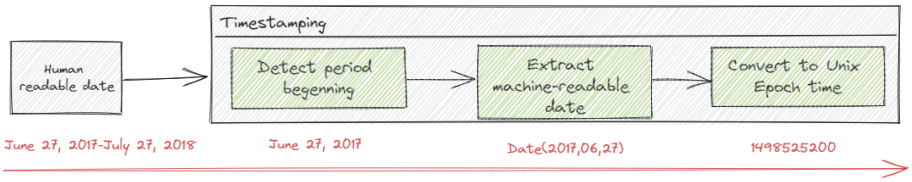

Figure 4.6: Timestamping module: below (red) shows an example of computing timestamp given a human-readable input period.

The LexNLP date extraction method allows two machine-readable formats to be detected: one for the beginning of the period and one for the end of the period. However, as, in this work, interaction events are just presumed to be link creations for the purpose of constructing the role predictor, the ends of periods were repudiated. In addition, because the time encoder in dynamic GNN models such as TGN requires an incremental and numerical representation of the dates such that $rep(2022, 3, 17) > rep(2022, 3, 16)$, all extracted dates should be transformed to **Unix Epoch Time** [52] and used as timestamps. During time conversion, it was discovered that certain periods have simply the beginning and ending years (e.g., 2012-2015). The sensitivity of the time dimension in this work necessitated the rejection of these role/period pairs in order to avoid imposing incorrect information, as a year encompasses a significant range in Epoch Time (from January to December, the range is $[0, 31449K]$). Fig. 4.7 illustrates the number of roles observed in

each time stamp. Each role held by a candidate at a particular time indicates an interaction event between the applicant and the role. Thus, as seen in Figure 4.7, the CV data mainly include more recent events than older ones.

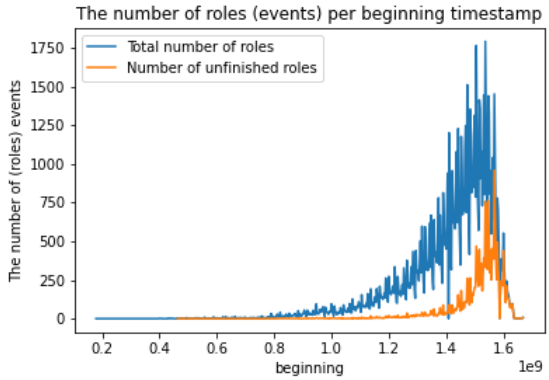

Figure 4.7: Distribution of the roles by Timestamp. Unfinished roles are the periods that are not ended (*e.g., the applicant still performs the role and writes June 2017-Present*).

Based on the timestamps obtained after preprocessing, cleaning, and Date Conversion, the CV data now have 102776 timestamped applicant-role pairs spread over 466 unique timestamps. Each timestamp, on average, includes 220.549 roles with a standard deviation of 345.082. Moreover, each applicant-role pair appears only once within a timestamp.

4.3 Graph Generation

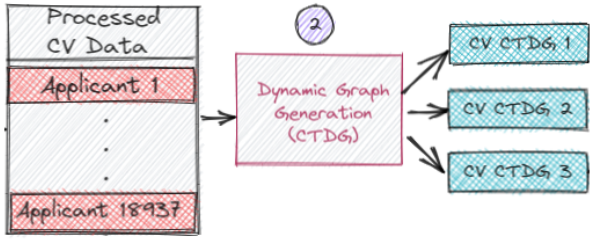

Figure 4.8: (Dynamic) Graph Generation Phase.

The second phase of our pipeline entails the generation of various possible bipartite graphs suitable for GNN models using CV Data that have been preprocessed. It is possible to obtain various graphs, and the models' performance may be sensitive to their input graphs. Consequently, it is worthwhile to explore various graph representations of the CV Data to determine the optimal representation for our final role prediction model.

4.3.1 CV Data Modeling

Borrowing the concept of data modeling from database design, an **entity-relationship (ER) model** [53] that describes the Applicant and Role relationship can be proposed, as illustrated in Fig. 4.9. This ER Data model depicts a general schema for modeling any (dynamic) bipartite graph used for role prediction based on the the available CV Data. Moreover, it suggests two entities[4] *Applicant*, *Role*, and one relationship *Take*. The *Applicant Entity* stores the applicant attributes. The *Role Entity* only maps the Role name to a unique ID to avoid redundancy. Other attributes may be considered for *Role*; however, only the Role name is currently practical given the existing CV data. Finally, the *Take relationship* describes a many-to-many relationship between the applicants and roles with respect to timestamps, which can be interpreted as *"an applicant can take many roles at any time, and a role can be also taken by different applicants at different times*.

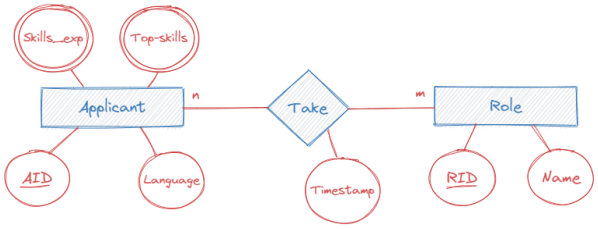

Figure 4.9: Conceptual Modeling of (dynamic) bipartite CV graph for role prediction using ER Diagram. *Double oval* represents a multi-valued attribute. *Underline* denotes primary key (Unique ID).

After processing the CV data as described in Section 4.2, it is possible to derive three distinct datasets based on the ER model in Fig. 4.9: one for storing application data comprising applicant attributes (*Table 4.2*), one for storing role data (*Table 4.3*), and one for capturing the applicant-role relationships represented by *Take Relationship*[5] (*Table 4.4*).

AID	Applicant unique ID
Language	Text (only one language provided)
Skills-exp	Set of skills applicant excels in (Multi-valued attribute)
Top-skills	Set of applicant's top skills (Multi-valued attribute)

Table 4.2: Elements of the Applicant dataset

Take dataset makes it possible to generate (dynamic) bipartite CV graphs. Any graph generated from this dataset will be a bipartite applicant-role graph. Each entry in the *Take dataset* represents an applicant-role interaction event. Consequently, the number of role nodes is proportionate to the number of unique roles in the *Role dataset*, and the number of applicant node is equal to the number of unique applicants in the *Applicant*

[4]Each rectangle represents an Entity in an ER diagram, and each diamond is a Relationship.

[5]When converting the conceptual ER diagram to a relational schema from which physical datasets (tables) are obtained, a many-to-many relationship is considered a dataset, making the relationship between the participating Datasets (entities).

RID	Role unique ID
Name	Text (Role Name)

Table 4.3: Elements of the Role dataset

AID	Applicant unique ID
RID	Role unique ID
Timestamp	a numerical representation of the beginning date of taking the role

Table 4.4: Elements of the Take dataset used for creating (dynamic) bipartite graphs.

dataset. Moreover, the *Applicant and Role datasets* are utilized only if features-enhanced graphs are of interest; otherwise, the *Take dataset* might be solely used to generate the (dynamic) CV bipartite graphs. Table 4.5 provides a statistical summary of the bipartite graph, which can be constructed from these three datasets.

Applicant Data	
The number of applicants	18486[6]
The number of unique languages	18
Number of unique Skills-exp	934
Number of unique Top-skills	5883
Role Data	
Number of roles	38938
Take Data	
Total Number of Interaction Events	102776
Mean Degree of role nodes (for Static Network)	2.566
Mean Degree of applicant nodes (for Static Network)	5.559
Number of unique timestamps	466
Mean (Stdv) number of Interaction Events per each timestamp	220.54 (345.08)

Table 4.5: Statistical Summary of a potential (dynamic) Bipartite graph that can be obtained by ER model in 4.9.

4.3.2 Feature Encoding

As described in Section 2.3, feature encoding is crucial when using a machine learning model because the machine only understands numbers. For instance, all text attributes must be transformed into numeric representations for the machine learning model to be able to ingest them. As with our Data model depicted in 4.9, the Applicant dataset includes Multi-valued attributes, such as Skills-exp and Top-skills, as well as a text attribute. In addition, the Role dataset contains only the Role name, which is text data. Therefore, in order to generate a features-enhanced (dynamic) bipartite graph that can be given

to models for role prediction, it is necessary to treat dataset non-numeric attributes and transform them into machine-understandable values. In this section, various attribute transformers that are utilized based on the attributes in the CV data model are discussed.

Multi-valued attributes

The Applicant dataset comprises two multi-valued attributes (Skills-exp and Top-skills) with each value being a string. Obviously, these attributes cannot be fed into the GNNs; therefore, they should be encoded. One common approach is to map each attribute into a (0,1)-feature vector x of dimensionality d where d is equal to the number of unique values observed for the attribute. In order to represent a multi-valued with the (0,1)-feature vector x, the unique values are ordered so that every value indexes a specific element in the vector. Hence, the values of the attribute are presented as entries in the vector x such that $x[i] = 1$ if the value i is observed in the attribute, $x[i] = 0$ otherwise. Although this approach seems logical in general, two drawbacks are associated with this, which make it impractical given our CV data. They are listed below:

- **Memory inefficiency:** The dimension d of the vector is proportional to the number of distinct values observed for the multi-valued attributes. If d is large, then a substantial amount of memory is required for the feature vectors of all applicants. For example, according to Table 4.5, there are 5883 distinct values for the attribute "Top-Skills" and 18486 candidates. For each applicant, a (0,1)-feature vector x must be stored; therefore, it blows up the memory with $18486 \times 5883 = 108,753,138$ elements to record Top-Skills attributes of all applicants, which explains the impracticality of this method.
- **Transductive Learning:**The (0,1)-feature vector prevents the model from generalizing to unseen nodes and renders it transductive, as the new unseen nodes may have attribute values not recorded in the training dataset.

Sentence embedding using the Sbert model[4] is recommended for memory-efficient feature encoding that supports inductive learning. For each candidate, a multi-valued attribute is first converted to a semicolon-separated text attribute that displays the attribute values as text (e.g., *"python;java;machine learning;cloud"*). Then, the new text attribute is embedded into a low-dimensional feature vector using the Sbert embedding model. Numerous pre-trained models based on the Sentence Bert architecture have been provided. Among these pre-trained models, the *"all-MiniLM-L6-v2"* model [7] is utilized in this work. The *"all-MiniLM-L6-v2"* model was fine-tuned using a large training dataset including more than one billion training pairs. This model is lightweight but precise, allowing it to provide a 384-dimensional embedding for multi-valued attributes. The beauty of this multi-valued encoding technique is that the resultant embedding encodes the similarity of the attributes so that two attributes that are closer in vector space have more similar values. For example, consider the semicolon-separated text attributes in Table 4.6. Similar attributes are observed to be closer in vector space after Sbert embedding, as seen in Fig. 4.10. For instance, the third and fourth attributes are associated with the Software Engineering profile. These two attributes share three explicitly similar values (**bold**) and one implicitly similar value (underline). As shown in Fig. 4.10, these two attributes are closer to one another and further away from the remainder of the attributes.

[7] https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2

1	'python;java;deep learning,cloud computing;'
2	'machine learning;python;deep learning;cloud'
3	**linux**;**java**;cloud;**oracle**;javascript;ajax;jquery'
4	'cloud computing;**java**;**linux**;mysql;**oracle**;web services'
5	'selenium;serenity;cucumber;business analyst'
6	'supply chain;planning;it;sql;electronic;visual studio;access;business analyst'

Table 4.6: Some multi-valued Skills-exp attributes converted to semicolon-separated text attributes

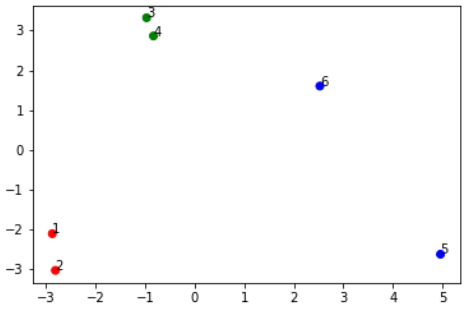

Figure 4.10: The embedding of the semicolon-separated text attributes in Table 4.6. The embeddings are reduced to a two-dimensional space using the PCA algorithm.

Text Attribute

Like multi-valued attributes, text attributes cannot be input into the model. Only the Language attribute is a text attribute in the Applicant dataset, deemed a nominal categorical variable. Thus, it could be encoded using one-hot encoding. Due to the fact that just 18 languages are accessible, as shown in Table 4.2, the model may be limited to transductive learning —- my mother tongue is Persian (a language spoken by 110 million people), yet it is absent from the CV Data. In order to facilitate inductive learning, the one-hot encoding is performed according to 72 of the most widely spoken languages on earth. It permits not only inductive learning but also retraining of the model once enough new applicant data has been collected.

In the Role dataset, just the role name, naturally a text attribute, is supplied as the role attribute. As roles are unique in this dataset, one-hot encoding can be utilized; nonetheless, it is not memory-efficient since there are 38938 unique roles. Therefore, the Sentence embedding using Sbert [4] with the pre-trained *"all-MiniLM-L6-v2"* model is applied because the role names are generally sentences, not single words. Furthermore, such an embedding helps preserve the role names' similarity and meaning when training a role prediction model because we have thousands of roles, and the chance of having overlapped roles is high. Take, for example, two existing roles in the Role dataset: *Java Software Engineer* and *Software Engineer*. As humans, we can simply realize that these two roles are almost identical— though the former is more specific. Therefore, if the role

prediction model understands and captures this similarity, it may predict a closer role when predicting the role for an unseen applicant whose actual role has not been seen during training (e.g., predicting *Software Engineer* while the actual role is *Software Engineer Technical Leader*) — in Section 4.6, it will be shown that the model could usually predict a role close to the actual role in the inductive test even if it has not been seen during training.

Summary

Following the encoding approaches proposed for the attributes of Applicant and Role datasets, the encoding details can be summarized in Tables 4.7 and 4.8.

Attribute	Encoding Technique	Dimension
Language	One-hot encoding	72-dimensional
Skills-exp	Embedded using Sbert [4]	384-dimensional
Top-skills	Embedded using Sbert [4]	384-dimensional

Table 4.7: Applicant dataset: Resulting Feature Encoding

Attribute	Encoding Technique	Dimension
Name	Embedded using Sbert [4]	384-dimensional

Table 4.8: Role dataset: Resulting Feature Encoding

4.3.3 Bipartite CV Graphs

According to Section 4.3.1, various temporal/static bipartite CV graphs can be formed using the *Take*, *Role*, and *Applicant* datasets with or without link/node features. This section will demonstrate various training graphs for temporal/static models. Even though the primary focus of this thesis is role prediction in temporal graphs, a static graph is provided as a baseline for training a static GNN. The topology and connectivity of all the graphs are identical, and they represent a bipartite graph; however, their node feature representations differ. These distinct graphs enable us to evaluate multiple models with diverse inputs and select the optimal dynamic graph representation, assisting the final dynamic role prediction model in achieving the desired performance — a good approximation of the optimality.

Static CV Graph: Baseline

The Static CV Graph (SCG) includes all the embedded features of both node types (Role and Applicant) but lacks the time dimension. The applicant's language, Skills-exp, and Top-skills are encoded for the Applicant dataset as explained in Tabel 4.7. Therefore, by concatenating all their encodings, the features of each applicant node would be a real-valued 840-dimensional vector. Moreover, only the role name is available for the Role dataset, encoded into a real-valued 384-dimensional vector, as explained in Table 4.8. Using the Take dataset, as shown in Fig 4.9, the bipartite static graph can be obtained if

the timestamps are eschewed. In summary, the SCG consists of 840-dimensional feature vectors for applicant nodes, 384-dimensional feature vectors for the role nodes, and the graph topology itself, emanated from the Take dataset.

Dynamic CV Graphs

In contrast to static models, the time dimension is fundamental in dynamic models. It is unnecessary to encode the timestamps further because all models automatically encode them during training, assuming they follow an incremental pattern, as discussed in Section 4.2.3. However, other essential components such as feature embeddings and **interaction event messages** (temporal link features) must be included. Four distinct dynamic graphs are proposed to examine the effect of various representations of the dynamic graphs on the performance of the dynamic models. In terms of topology, all graphs are identical, and only the feature representations are different.

- **Fully Features-Enhanced Dynamic Graph (FFDG):** In addition to time dimension and node features, interaction event messages are a vital component of temporal networks. These event messages must be unique; otherwise, they will interfere with the model training inadvertently. In the (Bipartite) Reddit Post graph [23], for instance, the text of each post is turned into a feature vector, and each feature vector acts as an event message between users and subreddits. However, only node features are given in the provided CV Data, and interaction event messages are absent. Therefore, the interaction event messages are artificially created by concatenating the applicant and role features, as explained in Tables 4.7 and 4.8. With this configuration, an interaction event message is a 1224-dimensional vector. This concatenation ensures uniqueness because each applicant starts performing a role at a specific timestamp. Furthermore, because adding node features leads to redundancy, low-dimensional zeros vectors are considered for the node features.

  Overall, the FFDG consists of zero features for nodes, 1224-dimensional feature vectors for interaction event messages, and the dynamic topology derived from the Take dataset with respect to timestamps.

- **Partially Features-Enhanced Dynamic Graph (PFDG):** PFDG is identical to FFDG and obtained in the same manner, except it does not take advantage of the Top-skill feature in the Applicant dataset for event message creation. FFDG is proposed to study the effect of the Top-skill attribute on the performance of the models as this attribute is suspected of adding noise to the model. Compared to Skills-exp, there are a large number of unique Top-skills; hence applicant nodes share fewer Top-skills. Consequently, it may cause the model difficulty in capturing the resemblance of the applicants' profiles.

  Altogether, PFDG consists of zero vector features for nodes of both types, 840-dimensional feature vectors for the event messages attained by concatenating applicants' language, Skills-exp, and role name features, as well as the dynamic topology derived from the Take dataset with respect to timestamps.

- **Non-Feature Dynamic Graph (NFDG)**: The impact of the features on the model's performance is worth studying. One way to do this investigation is to build a dynamic graph that does not benefit from any features and event messages. NFDG draws only the topology of the graph that evolves over time.

4.4 Best Graph Selection

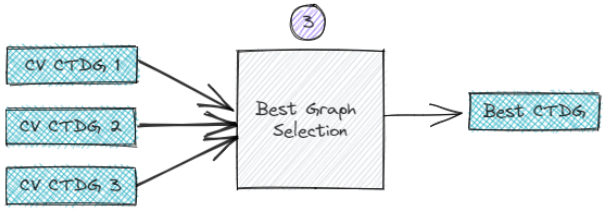

Figure 4.11: Best Graph Selection Phase.

The best Graph Selection phase aims to identify the best dynamic Graph representation of the CV Data by training well-known dynamic Models on the dynamic graphs in Section 4.4.2. Given the constructed dynamic graphs in section 4.4.2, the job of role prediction is reduced to predicting the links between applicants and roles on bipartite graphs. The optimal graph should be chosen based on the model's overall performance with recommendation metrics such as $Recall@K$; however, this is computationally expensive due to the number of computations involved; Cross-validation, predicting the role for an unseen node given his/her history, resetting the model, etc. For instance, a single round of Cross-validation using $Recall@K$ requires approximately eight hours on the machine allocated by Euranova. In contrast, each round takes, on average, ninety minutes if average precision is concerned. Therefore, for the sake of simplicity and efficiency, the optimal dynamic graph will be chosen based on the average precision of the models— though it just gives an estimation of the model performance as a role predictor.

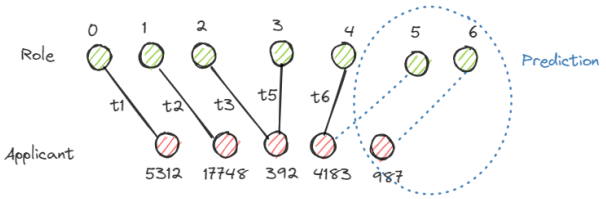

Figure 4.12: Model Evaluation using Average Precision. In Average precision, the existence and non-existence of a (temporal) link using negative sampling are tested like binary classification.

4.4.1 Static Model: GAT Encoder-Decoder (GATED)

As described in Section 3.1, the link prediction model should comprise an encoder and decoder framework. The static encoder is made up of many GAT Convolutional (explained in Section 3.7) and dropout layers. The decoder consists of two linears and dropout layers to estimate the likelihood of linkages between applicant and role nodes. The model is trained using positive and negative link samples, with Binary Cross Entropy serving as the loss function. In addition, the hyper-parameters for the GATConv, Dropout, and Linear layers were chosen using the manual knob-tuning technique using validation samples, and

the optimal model was chosen using the early-stopping technique to prevent overfitting. Notice that the Standard Message Passing technique cannot be applied to bipartite graphs in the encoder because nodes of various types possess distinct features that the same functions cannot process. PyGeometric [54], a python GNN framework, offers an automated method for converting homogeneous GNN models/layers to heterogeneous GNN models/layers. The conversion can only be performed by changing the encoder using the **"to heter"** transformer, which duplicates the encoder's message functions to operate independently for each type. Fig. 4.13 depicts the schema of GATED, which employs the **"to heter"** transformer to construct the GAT-based link predictor for the SCGE bipartite graph.

Notably, static models are not the primary focus of this thesis; however, one static model serves as a baseline for evaluating the performance of dynamic models.

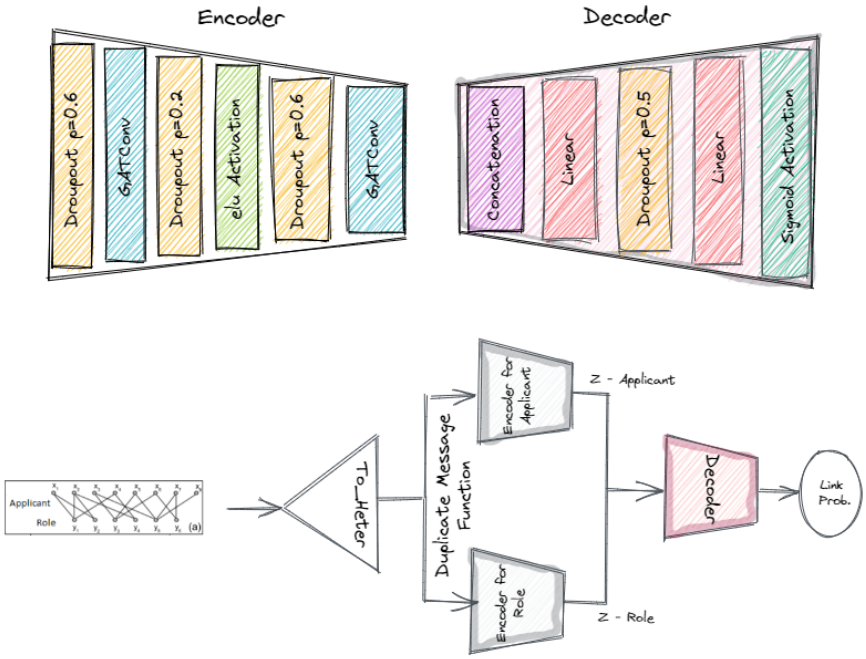

Figure 4.13: Role Prediction GAT Encoder-Decoder.

4.4.2 Dynamic Models

As the dynamic encoder, the popular TGN [2], JODIE [23], and Dyrep [48] are utilized. All these dynamic models have been found to outperform static models. For all these dynamic encoders, an MLP with two layers is used as decoder. The learning rate is set to be 0.0001. Moreover, Dropout probability, Node Embedding dimension, and Time Embedding dimension if applied are set to be 0.1, 100, and 100 for all dynamic models.

TGN Model

TGN has multiple configurations; TGN-attn is chosen since it has been found to be superior to the other configurations. TGN-attn employs a Graph Attention with two heads, one layer, and ten neighbors for embedding. For TGN-attn, the configuration and hyper-parameters are chosen as in Table 4.9:

Memory Updater	Gated Recurrent Unit (GRU)
Message Aggregator	Most Recent Message in the batch(Last)
Message Function	Concatenation of the inputs
Memory Dimension	172
Node Embedding Dimension	100
Time Embedding Dimension	100
Dropout probability	0.1

Table 4.9: Selected hyper-parameters and configuration for TGN

JODIE Model

The JODIE's memory updater is a Recurrent Neural Network, and embedding is performed using the Time projection approach, as described in [23] or Section 3.3. The message function is merely the concatenation of the inputs; no complicated method is employed.

Dyrep Model

Like JODIE, Dyrep utilizes an RNN for its memory updater. The memory is directly used as embedding. Moreover, it uses graph attention to add a summary of the destination node neighborhood to the message function.

4.4.3 Model and Graph Performance

This section presents the experimental results obtained through running the models explained in Section 4.4.2 on the CV graphs in Section 4.3.3. For all experiments and graphs, the batch size is set to 150, and the graphs are split into 70%-15%-15% chronological splits.

	FFDG		PFDG		NFDG		Best
	Trans.	Induct.	Trans.	Induct.	Trans.	Induct.	
GATED	-	-	-	-	-	-	SCG
Dyrep	0.9337	0.8208	0.9381	0.8288	0.9270	0.8098	PFDG
JODIE	0.9328	0.8228	0.9380	0.8333	0.9274	**0.8177**	PFDG
TGN-attn	**0.9580**	**0.8251**	**0.9613**	**0.8343**	**0.9500**	0.8100	PFDG

Table 4.10: Average Precision for future role prediction task in transductive (Trans.) and inductive (Induc.) settings. **Bold** is best and Underline is the second best. *For SCG graph and GATED model, the average precision in the transductive test is 0.691, and in the inductive test is 0.6313.* The results are the average of five runs.

A cursory look at 4.10 reveals that TGN-attn outperforms all other models in all graphs in both settings except in NFDG where the average precision in the inductive setting is negligibly lower than Jodie. However, the difference is mostly observed in the transudative setting as all other models could achieve an average precision below 0.94 in all graphs, while this average precision for TGN-attn is always equal or larger than 0.9500.

Table 4.10 also unveils that adding more features to the dynamic CV graph does not necessarily result in a more accurate role prediction model, as the PFDG graph consistently performs better. The superior performance achieved by the PFDG graph suggests that Top-skills features not only do not raise the average accuracy in both settings but also somewhat decrease it— Top-skills features are not added to PFDG. In addition, the NFDG graph surprisingly enables models to train a relatively accurate role predictor without any features. In other words, the time dimension appears to be the most crucial aspect in the dynamic CV graphs, which is sufficient to overwhelm the static model.

Altogether, due to the higher average precision achieved by PFDG — *though it is not very significant*— and the fact that it is more memory efficient than FFDG, the best dynamic graph representation of the CV Data is chosen to be PFDG. Therefore, in the remaining of the pipeline, whenever the term dynamic graph is mentioned, it refers to as a **PFDG instance** of the CV graph unless it is explicitly mentioned.

4.5 Best Model Selection

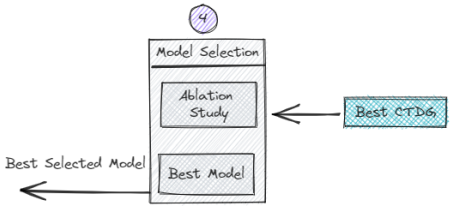

Figure 4.14: Best Model Selection Phase.

In this phase, many instances of the TGN model on PFDG are studied. The ablation study in the original publication [2] inspires this analysis. The objective of this analysis is to determine how adding, removing, or altering a component in TGN affects performance in terms of average precision in both transductive and inductive tests. Moreover, it aids in determining the optimal TGN configuration for the role prediction given the PFDG Graph. It is important to note that the findings may be suboptimal due to the fact that only one representation of the CV Graph is used and several hyper-parameters are involved; yet, the results can provide a reasonable approximation of the ideal configuration of TGN, which can be used in the final role prediction model.

4.5.1 TGN Model Instances

The TGN instances listed in Table 4.9 are utilized for the ablation study. All the TGN instances in Tabel 4.11 except TGN-no-mem that does not have memory are merely different in terms of Embedding Module.

For all TGN based-models except TGN-no-mem, the configuration and hyper-parameters are chosen as in Table 4.9. Notice that TGN-no-mem does not benefit from any memory.

Name	Embedding Module
TGN-attn	Attention with two heads, one layer, and 10 neighbs.
TGN-2l	Attention with two heads, two layers, and 10 neighbs.
TGN-no-mem	Attention with two heads, one layer, and 10 neighbs, no memory.
TGN-time	Time projection.
TGN-ID	Use the memory directly as the node embedding.
TGN-Sum	Sum graph convolution using 1 layer and 10.

Table 4.11: Different TGN models with various embedding modules.

Therefore, it lacks Memory, Memory updater, Message Function, and Message Aggregator; nevertheless, its node embedding dimension and dropout probability are similarly set to 100 and 0.1, respectively.

Note that, similar to Section 4.4.3, the batch size is set to 150, and the graphs are split into 70%-15%-15% chronological splits.

4.5.2 Results

Fig 4.17 illustrates that TGN models with more complicated embedding modules outperform other models. For example, TGN-2l with two graph attention convolutional layers has the highest average precision in transductive and inductive scenarios. Surprisingly, TGN-no-mem performs well in the transductive environment; however, its sub-0.8 inductive precision indicates the significance of the memory module, as all other models are able to achieve an average precision of 0.82 or higher in the inductive setting. This in-depth study suggests employing additional attention graph convolutional layers to create a more trustworthy TGN model for role prediction, which optimally performs on observed and unseen nodes. **TGN-2L** is then the proposed dynamic model that is expected to function best with the PFDG representation of CV Data.

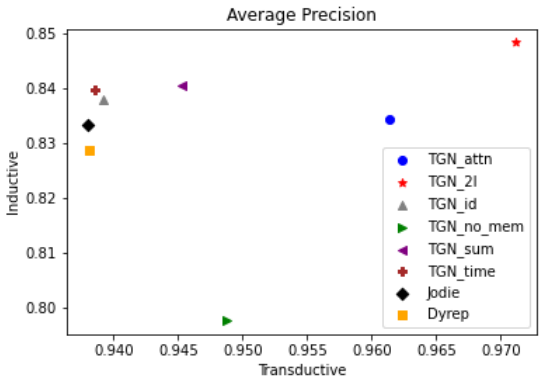

Figure 4.15: Average Precisions in Inductive and Transductive settings. The results are the average of five runs.

4.6 Evaluation as Top-k Role Recommendation

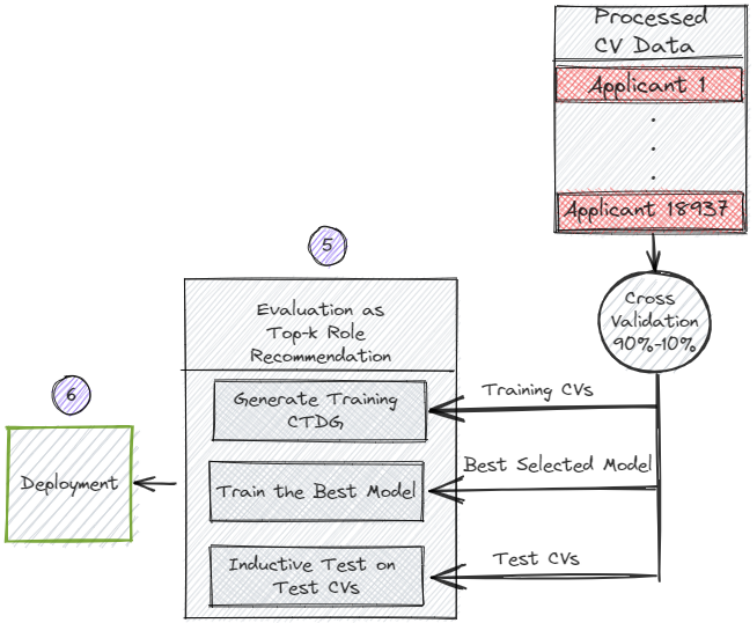

Figure 4.16: Evaluation as Top-k Role Recommendation System.

So far, all evaluations have been conducted as a role prediction task that can be cast as link prediction in which the average precision is calculated based on a set of negative and positive samples, such that the problem can be considered a classification problem — to distinguish existent links from the non-existent ones. Moreover, the PFDG representation of the CV graph and TGN-2l model have been nominated based on the average precision evaluation performed in Section 4.4.3 since it could simplify the pipeline's model selection process, which could be a bottleneck.

Due to the size of the accessible item set, it is typically impossible for many recommendation systems to predict or recommend only one item to a user. Predicting one actual item out of thousands of items in the system requires a model with a precision close to 1, computed over all possible items, not just a set of negative and positive samples. Therefore, a typical recommendation system suggests that top-k items should be retrieved instead of only one item. A top-k recommendation system aims to recommend as many positive items as possible in the top-k recommended items.

In this section, the proposed role prediction model (PFDG+TGN-2l) is examined as a top-k role recommendation system, where it recommends top-k most potential roles as the subsequent role for an applicant given the applicant's past experiences.

In order to conduct the experiment in this phase of the pipeline, as depicted in Fig. 4.15, the Cross-validation evaluation approach is used to obtain a trustworthy evaluation of the model using various training and test datasets. This step includes three inputs resulting from the previous phases: Training processed CV (90% of the CV datasets[8]) dataset, Test

[8]16637 training applicants (roughly 92817 interaction events). %80 of training data are chosen

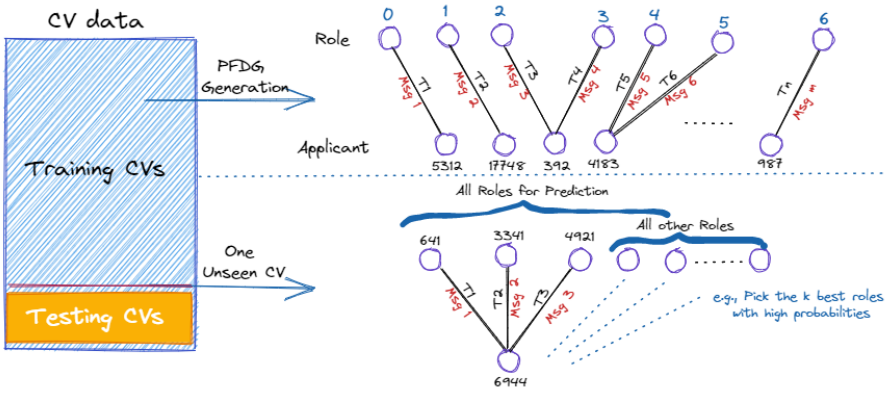

Figure 4.17: One round of the Cross-validation for top-k recommendation evaluation

processed CV dataset (10% of the CV dataset [9]), and the TGN-2l as the best model found on the PFDG representation of the CV data. At each Cross-validation round, a PFDG representation of the training dataset (training PFDG) is generated, as this is expected to be the most accurate representation. Then, a TGN-2l model is trained on the training PFDG with the same hyper-parameters[10] used in the experiments in section 4.5.1. After training the model, each applicant's CV from the test data is selected for the model to be evaluated on. The last applicant's role is selected as the ground-truth role for evaluation. The model first ingests the applicant's previous roles (except the last roles) as interaction events between the applicant and roles, then updates its states, and finally computes an embedding for the applicant. Based on the computed embedding, the probabilities of the links between the applicant and all the roles are calculated, helping pick the top-k most probable roles for the applicant. A top-k evaluation such as $Recall@K$—discussed later—can then be applied according to the retrieved top-k roles. Fig. 4.16 illustrates one round of the cross-validation procedure for evaluating the model, whereas Algorithm 1 provides the pseudocode for the entire top-k evaluation procedure.

4.6.1 Evaluation Metrics

Considering Algorithm 1 and Fig. 4.6, the suggested top-k role recommender can be evaluated using the cross-vadiation approach. However, evaluation requires a top-k recommendation metric. This section presents two metrics, $Recall@K$ and $Cosine-Similarity@k$ *(defined in this thesis)*, which are used to evaluate the top-k role recommender.

Recall@K

$Recall@K$ computes the proportion of pertinent items among the top-k recommended items. Let P_u be a set of positive items the user will interact with in the future, and R_u be a set of k items recommended by a model. Therefore, the $Recall@K$ is calculated by

for training, %15 for transductive test, %5 for inductive test. These %20 validation data are used for checking the correctness of the model and overfitting (not related to top-k metrics.)

[9] 1848 test applicants

[10] The only difference is that the batch size is set to be 200 for the sake of efficiency.

Algorithm 1 Top-k Role Recommendation Evaluation

Input: $CVDATA$ (Processed CV Data), k (k Best prediction)
Output: $Meam\ score$ for an Evaluation Metric (e.g, $Recall@K$)

$CV \leftarrow$ Split CVDATA into 10 folds $F = \{f_1, f_2, ..., f_{10}\}$
$Meam\_score \leftarrow 0$
for $m \leftarrow 1\ to\ 10$ **do**
  $T_m \leftarrow \{F - f_m\}$ ▷ Use f_m as a test CVs and T_m as training CVs
  $PFDG_m \leftarrow Create\_PFDG(T_m)$ ▷ Create the train PFDG based on T_m
  $Role\_Nodes \leftarrow$ pick all role nodes in $PFDG_m$
  $R_{no} \leftarrow |Role\_Nodes|$
  $M \leftarrow Train(TGN\_2l, T_m)$ ▷ Train TGN-2l model on T_m
  for each $cv \in f_m$ **do**
    $AID \leftarrow create\_id(cv)$ ▷ Create an ID for the applicant
    $Probs \leftarrow \mathbb{R}^{|R_{no}|}$ ▷ To keep the probability of the links (AID, $role_i$)
    $M_i \leftarrow M$ ▷ To avoid updating model for next cv
    ▷ Order roles in cv chronologically
    $\{role_{t_1}, role_{t_2}, ..., role_{t_n}\} \leftarrow Order\_roles(cv)$
    $Experiences \leftarrow \{role_{t_1}, role_{t_2}, ..., role_{t_{n-1}}\}$
    $Role_{last} \leftarrow role_{t_n}$ ▷ The ground-truth role for evaluation
    $MSGs \leftarrow Compute\_Message(AID, Experiences)$
    for each $role_i \in Experiences\ and\ msg_i \in MSGs$ **do**
      $M_i \leftarrow Insert(M_i, event(AID, role_i, msg_i))$ ▷ New event to M_i
    end for
    for each $role_i \in Role\_Nodes$ **do**
      $Probs_i \leftarrow Predict\_Link(M_i, AID, role_i)$
    end for
    $Top_k \leftarrow Pick\_best(Role\_Nodes, Probs, k)$ ▷ Pick k best prediction
    $score+ \leftarrow Evaluation\_Metric(Top_k, Role_{last})$ ▷ metric as $Recall@K$
  end for
  $Meam\_score+ \leftarrow score/|f_n|$
end for
$Meam\_score \leftarrow Meam\_score/10$
return $Meam\_score$

ID	Test Mode	Ground-truth Role	Predicted Role	CM
1	Unseen	software engineer technical leader	software engineer	0.72
2	**Seen**	**devops cloud engineer**	**devops engineer**	**0.89**
3	Unseen	c python	software engineer	0.23
4	Seen	business development leader	business developer	0.7
5	**Seen**	**sales business developer**	**business developer**	**0.86**

Table 4.12: Some predictions made by the role recommender. Although they are all deemed wrong in a binary manner, many of them are pretty close in their meanings. CM is Cosine Similarity computed after embedding the true and predicted roles using Sbert [4]. **Unseen** means the true role has not been seen during training, and **Seen** indicates the true role has been seen during training.

$$Recall@K = \frac{|R_u \cap P_u|}{|P_u|} \tag{4.1}$$

In the top-k role recommendation, $|P_u|$ equals one because, at any time, only one positive example is available for any applicant—only the last role is chosen as the ground truth. Accordingly, the numerator in Eq. 4.2 can only take on either zero or one. Therefore, the $Recall@K$ metric for the top-k role recommendation is a binary evaluation metric that checks the ground-truth role absence or presence in the top-k recommended roles. This binary metric results in a lower $Recall@K$ value for the top-k role recommender than other top-k recommenders with many ground-truth items. When there are more ground-truth items, the numerator is expected to be higher than zero in many cases.

Cosine-Similarity@k

As explained in Eq. 4.2, the $Recall@K$ metric is expected to be low in the top-k role recommendation due to the binary nature of the $Recall@K$ in this recommendation system. However, the chances are that the recommender gives a list of top-k roles such that at least one of the roles is close to the ground-truth role in terms of meaning, as there are 38938 role nodes obtained from the IT industry, and consequently, many of them may overlap (e.g., many similar roles are written differently). Take, for example, some predictions in Table 4.6.1 made by the proposed role recommender, which are evaluated to be incorrect in terms of $Recall@K$. As seen in Table 4.6.1, even though they are deemed incorrect by $Recall@K$, leading to zero $Recall@K$, many of the predictions are rather near to the ground-truth roles, indicating the model's capability of recommending similar roles. Therefore, it necessitates a metric that can reflect the similarity of the predicted and true roles in order to acquire a fair top-k role recommendation assessment.

Definition 1 *$Cosine - Similarity@K$ is the highest cosine similarity computed between the ground-truth role and the top-k recommended roles.*

Seen vs. Total Results

When predicting the role of an applicant, it is possible that the ground-truth role has not been seen during training. Therefore, as both applicant node and true role have not

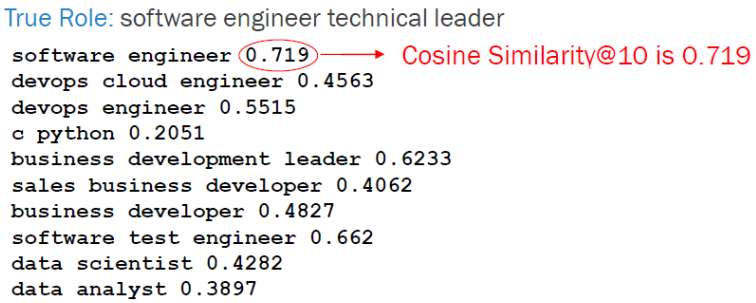

Figure 4.18: An example of $Cosine - Similarity@K$ with $K = 10$, for the top-10 recommended roles.

been seen during training, the model definitely fails to predict the true role, leading to a zero $recall@k$, even though the model is inductive— it will be shown in Section 4.6.2. The reason is that, unlike static models, all the nodes are not inserted at once. In the continuous dynamic models, the events emerge over time, and if the ground-truth role has not been involved in any events, the model does not create any memory for it. Ergo, how would it be possible for the model to predict something that it has ever seen? Hence, the results can be presented according to three test modes, which are as follows:

- **Seen:** the true role has been seen during training; thus, the model should predict the true role.
- **Unseen:** the true role has not been seen during training; thus, the model fails to predict the true role.
- **Total:** Seen and Unseen test modes are considered.

However, in Section 4.6.2, only Seen and Total test modes are utilized.

4.6.2 Evaluation Results

For the purpose of evaluating the performance of the proposed model using Algorithm 1, the following models are chosen for evaluation.

- neg_sample_i : The proposed role recommendation model (PFDG+TGN-2L) with $i \in \{1, 20, 100\}$ negative samples, denoted by neg_sample_i.
- **Baseline 1:** Predict the current role of an applicant as his/her next role. It is worth noting that because the last role of supplicants is predicted as the next role, the choice of K does not result in different evaluation metric score. In other words, the $Recall@K$ and $Cosine - Similarity@K$ for Baseline 1 are equal for every value of K.
- **Baseline 2:** Randomly select k roles with probabilities proportional to their frequency.

$$P_{role_i} = \frac{\#\ of\ observations\ of role_i}{\sum_j \#\ of\ observations\ of\ role_j} \tag{4.2}$$

To pick the best random prediction, Ten Samplings are performed.

Results

First and foremost, it should be remarked that all the results are the average scores obtained by the cross-validation test, and the bar on the lines indicates the standard deviation. Moreover, the $Recall@K$ and $Cosine - Similarity@K$ for the Baseline 1 for any K are 0.006 (12.4 correct answers) and 0.43, respectively. This result indicates that almost all applicants take a different role from their previous roles, making the role prediction task more challenging. Finally, while predicting the next roles of the test applicants using the top-k recommender, an average of 23 percent of the ground-truth roles are not seen during training, which results in an average of 1425 Tests in the **Seen** test mode.

At first glance, all Figures 4.19,4.20,4.21, and 4.22 reveal that $Recall@K$ and $Cosine-Similary@k$ both monotonically increase as K grows. Furthermore, it is observable that the top-k role recommender with higher negative samples for training increases the performance in the $Recall@K$ but does lead to better performance in $Cosine - Similary@k$. However, more negative samples do not necessarily help train a better model as a neg-sample-20 and neg-sample-100 produce the same results in both seen and unseen modes. Altogether, it can be deduced that 20 negative samples are enough for training.

Because there are 38938 roles to be predicted and only one ground-truth role is provided, the $Recall@10 = 0.28$ in the seen test mode implies the prediction power of the top-10 role recommender —28% of the top-k recommended roles contain the ground-truth role. Furthermore, although $K = 35$ may appear to be a high number, K can be chosen for hundreds in practice [55]. Thus, $Recall@35$ represents a reasonable performance given the complexity of the data.

As described in Section 4.6.1, it is evident that the $Recall@k$ for any k is lower in the Total mode due to the contribution of zero scores to the overall score. In such cases, it may be prudent to investigate $Cosine - Similarity@K$ in the total mode to evaluate the performance of the model in retrieving comparable but not identical ground-truth roles.

As seen in Figures 4.21 and 4.22, the $Cosine - similarity@k$ for $k = 10$ is around 0.70 for both the Seen and Total modes. By taking Table 4.6.1 as a reference, it is possible to determine that the similarity of 0.7 corresponds to two roles whose meanings are quite similar. Consequently, the model performance in terms of $Cosine - similarity@k$ for $k \geq 10$ suggests that at least one of the recovered top-k roles may be similar to the ground-truth roles, even if it was not utterly observed during training.

Overall, the experimental findings produced by Algorithm 1 reveal that the top-k role recommender developed on the basis of a continuous dynamic model demonstrates acceptable performance in terms of both metrics for $k \geq 10$.

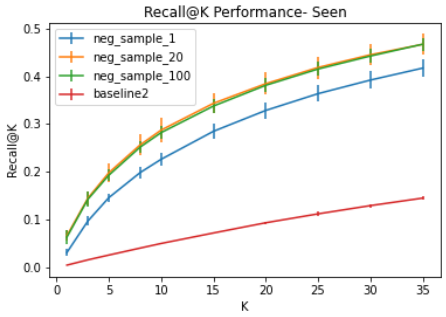

Figure 4.19: $Recall@K$ - Seen

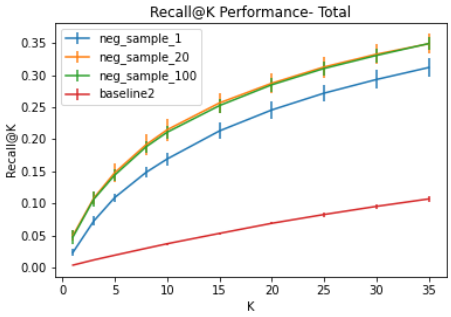

Figure 4.20: $Recall@K$- Total

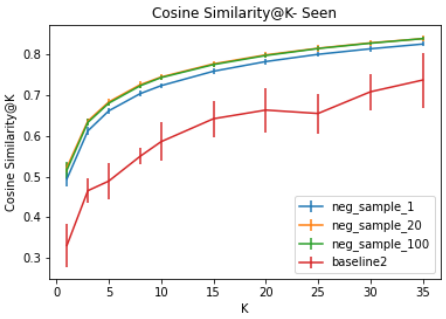

Figure 4.21: Cosine-Similarity@K- Seen

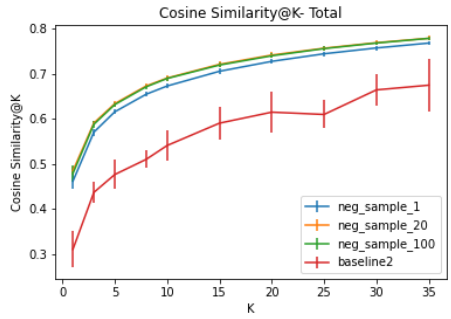

Figure 4.22: Cosine-Similarity@K - Total

Chapter 5

Conclusion

In this thesis, a new framework for the top-k role recommendation system has been proposed, which benefits from the time dimension. Multiple research on the CV Data and various dynamic graph representations of the CV Data have demonstrated that the PDFG representation yields the most promising results on several dynamic models. Then Based on this representation, the best model, TGN-2l, was chosen, which ultimately was applied in the top-k recommender. According to the experimental results measured by the $Recall@k$ and $Cosine - Similary@k$, the proposed top-k recommender has been found to recommend a list of top-k roles that are highly resemblant for $k \geq 10$ even when the ground-truth roles are not retrieved. It comes from the fact that the $Cosine-Similary@k$ for $k > 10$ is above 0.7, indicating high similarity.

Although the model has shown promising results, it might be improved in several ways:

- The use of the TGN enhancement is discussed in Appendix A. For example, APAN in Fig. A.1 may help have faster role recommender because TGN is a bit slow in deployment.
- Providing a more clean CV Data set with fewer available roles. In the dataset, it has been observed that many roles overlap, making the prediction harder.

Bibliography

[1] Jacob Levy Moreno. Who shall survive. In *Who shall survive:A New Approach to the Problem of Human Interrelations*. Nervous and Mental Disease Publishing Co., 1934.

[2] Emanuele Rossi, Ben Chamberlain, Fabrizio Frasca, Davide Eynard, Federico Monti, and Michael Bronstein. Temporal graph networks for deep learning on dynamic graphs. *arXiv preprint arXiv:2006.10637*, 2020.

[3] Xuhong Wang, Ding Lyu, Mengjian Li, Yang Xia, Qi Yang, Xinwen Wang, Xinguang Wang, Ping Cui, Yupu Yang, Bowen Sun, et al. Apan: Asynchronous propagation attention network for real-time temporal graph embedding. In *Proceedings of the 2021 International Conference on Management of Data*, pages 2628–2638, 2021.

[4] Nils Reimers and Iryna Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks. *arXiv preprint arXiv:1908.10084*, 2019.

[5] Shaha T Al-Otaibi and Mourad Ykhlef. A survey of job recommender systems. *International Journal of Physical Sciences*, 7(29):5127–5142, 2012.

[6] Supreet Kaur and Sonia Sharma. Review on security techniques using cloud computing.

[7] Tin Van Huynh, Kiet Van Nguyen, Ngan Luu-Thuy Nguyen, and Anh Gia-Tuan Nguyen. Job prediction: From deep neural network models to applications. In *2020 RIVF International Conference on Computing and Communication Technologies (RIVF)*, pages 1–6. IEEE, 2020.

[8] Pengyang Wang, Yingtong Dou, and Yang Xin. The analysis and design of the job recommendation model based on gbrt and time factors. In *2016 IEEE International Conference on Knowledge Engineering and Applications (ICKEA)*, pages 29–35. IEEE, 2016.

[9] Mark EJ Newman. The structure and function of complex networks. *SIAM review*, 45(2):167–256, 2003.

[10] Markus Brede. Networks—an introduction. mark e. j. newman. (2010, oxford university press.) $65.38, £35.96 (hardcover), 772 pages. isbn-978-0-19-920665-0. *Artificial Life*, 18:241–242, 2012.

[11] C.A. Myers and G.P. Shultz. The dynamics of a labor market. New York: Prentice Hall, 1951.

[12] Thomas J Fararo and Morris H Sunshine. *A study of a biased friendship net*. Youth Development Center, Syracuse University, 1964.

[13] Anatol Rapoport and William J. Horvath. A study of a large sociogram. *Behavioral Science*, 6(4):279–291, 1961.

[14] Joseph Galaskiewicz. *Social Organization of an Urban Grants Economy*. Elsevier, 1985.

[15] Jeffrey Travers and Stanley Milgram. An experimental study of the small world problem. In *Social networks*, pages 179–197. Elsevier, 1977.

[16] Aya Zaki, Mahmoud Attia, Doaa Hegazy, and Safaa Amin. Comprehensive survey on dynamic graph models. *International Journal of Advanced Computer Science and Applications*, 7(2), 2016.

[17] William L Hamilton. Graph representation learning. *Synthesis Lectures on Artifical Intelligence and Machine Learning*, 14(3):1–7, 2020.

[18] Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Galligher, and Tina Eliassi-Rad. Collective classification in network data. *AI magazine*, 29(3):93–93, 2008.

[19] Joakim Skarding, Bogdan Gabrys, and Katarzyna Musial. Foundations and modeling of dynamic networks using dynamic graph neural networks: A survey. *IEEE Access*, 9:79143–79168, 2021.

[20] Jinyin Chen, Jian Zhang, Xuanheng Xu, Chenbo Fu, Dan Zhang, Qingpeng Zhang, and Qi Xuan. E-lstm-d: A deep learning framework for dynamic network link prediction. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 51(6):3699–3712, 2019.

[21] Kuansan Wang, Zhihong Shen, Chiyuan Huang, Chieh-Han Wu, Yuxiao Dong, and Anshul Kanakia. Microsoft Academic Graph: When experts are not enough. *Quantitative Science Studies*, 1(1):396–413, 02 2020.

[22] Yinwei Wei, Xiang Wang, Liqiang Nie, Xiangnan He, Richang Hong, and Tat-Seng Chua. Mmgcn: Multi-modal graph convolution network for personalized recommendation of micro-video. In *Proceedings of the 27th ACM International Conference on Multimedia*, pages 1437–1445, 2019.

[23] 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*, pages 1269–1278, 2019.

[24] Hanjun Dai, Yichen Wang, Rakshit Trivedi, and Le Song. Deep coevolutionary network: Embedding user and item features for recommendation. *arXiv preprint arXiv:1609.03675*, 2016.

[25] Di Jin, Zhizhi Yu, Pengfei Jiao, Shirui Pan, Dongxiao He, Jia Wu, Philip Yu, and Weixiong Zhang. A survey of community detection approaches: From statistical modeling to deep learning. *IEEE Transactions on Knowledge and Data Engineering*, 2021.

[26] Giulio Rossetti and Rémy Cazabet. Community discovery in dynamic networks: a survey. *ACM computing surveys (CSUR)*, 51(2):1–37, 2018.

[27] Jure Leskovec, Kevin J Lang, Anirban Dasgupta, and Michael W Mahoney. Community structure in large networks: Natural cluster sizes and the absence of large well-defined clusters. *Internet Mathematics*, 6(1):29–123, 2009.

[28] Aaron Clauset, Mark EJ Newman, and Cristopher Moore. Finding community structure in very large networks. *Physical review E*, 70(6):066111, 2004.

[29] Valdis Krebs. Uncloaking terrorist networks. *First Monday*, 7(4), Apr. 2002.

[30] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. *Advances in neural information processing systems*, 30, 2017.

[31] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

[32] Liang Yao, Yin Zhang, Baogang Wei, Zhe Jin, Rui Zhang, Yangyang Zhang, and Qinfei Chen. Incorporating knowledge graph embeddings into topic modeling. In *Thirty-first AAAI conference on artificial intelligence*, 2017.

[33] 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*, pages 855–864, 2016.

[34] Da Xu, Chuanwei Ruan, Evren Korpeoglu, Sushant Kumar, and Kannan Achan. Inductive representation learning on temporal graphs. *arXiv preprint arXiv:2002.07962*, 2020.

[35] Thomas N Kipf and Max Welling. Variational graph auto-encoders. *arXiv preprint arXiv:1611.07308*, 2016.

[36] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[37] 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)*, pages 1532–1543, 2014.

[38] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[39] Quoc Le and Tomas Mikolov. Distributed representations of sentences and documents. In *International conference on machine learning*, pages 1188–1196. PMLR, 2014.

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

[41] Claudio DT Barros, Matheus RF Mendonça, Alex B Vieira, and Artur Ziviani. A survey on embedding dynamic graphs. *ACM Computing Surveys (CSUR)*, 55(1):1–37, 2021.

[42] William L Hamilton, Rex Ying, and Jure Leskovec. Representation learning on graphs: Methods and applications. *arXiv preprint arXiv:1709.05584*, 2017.

[43] Seyed Mehran Kazemi, Rishab Goel, Kshitij Jain, Ivan Kobyzev, Akshay Sethi, Peter Forsyth, and Pascal Poupart. Representation learning for dynamic graphs: A survey. *J. Mach. Learn. Res.*, 21(70):1–73, 2020.

[44] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[45] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S Yu Philip. A comprehensive survey on graph neural networks. *IEEE transactions on neural networks and learning systems*, 32(1):4–24, 2020.

[46] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE transactions on neural networks*, 20(1):61–80, 2008.

[47] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

[48] Rakshit Trivedi, Mehrdad Farajtabar, Prasenjeet Biswal, and Hongyuan Zha. Dyrep: Learning representations over dynamic graphs. In *International conference on learning representations*, 2019.

[49] Ilya Makarov, Andrey Savchenko, Arseny Korovko, Leonid Sherstyuk, Nikita Severin, Aleksandr Mikheev, and Dmitrii Babaev. Temporal graph network embedding with causal anonymous walks representations. *arXiv preprint arXiv:2108.08754*, 2021.

[50] Yiwei Wang, Yujun Cai, Yuxuan Liang, Henghui Ding, Changhu Wang, Siddharth Bhatia, and Bryan Hooi. Adaptive data augmentation on temporal graphs. *Advances in Neural Information Processing Systems*, 34, 2021.

[51] Michael J Bommarito II, Daniel Martin Katz, and Eric M Detterman. Lexnlp: Natural language processing and information extraction for legal and regulatory texts. In *Research Handbook on Big Data Law*. Edward Elgar Publishing, 2021.

[52] Wikipedia. Unix time — Wikipedia, the free encyclopedia, 2022. [Online; accessed 3-June-2022].

[53] Peter Chen. *Entity-Relationship Modeling: Historical Events, Future Trends, and Lessons Learned*, pages 296–310. Springer Berlin Heidelberg, Berlin, Heidelberg, 2002.

[54] Matthias Fey and Jan Eric Lenssen. Fast graph representation learning with pytorch geometric. *arXiv preprint arXiv:1903.02428*, 2019.

[55] Xiwang Yang, Harald Steck, Yang Guo, and Yong Liu. On top-k recommendation using social networks. In *Proceedings of the sixth ACM conference on Recommender systems*, pages 67–74, 2012.

Appendices

Appendix A

TGN Enhancements

A.1 Asynchronous Propagation Attention Network

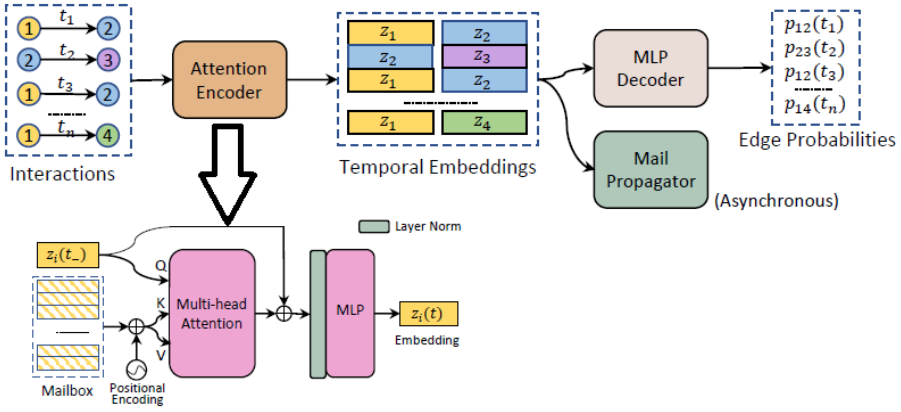

Figure A.1: APAN Framework[3].

Current dynamic models such as TGN and TGAT suffer from excessive latency on the online inference since these methods need visiting nodes' k-hop temporal neighbors before generating the node embeddings of nodes participating in interactions.

In [3], the authors discuss the significance of online inference latency in real-world applications such as Online payment apps, where embezzlement and fraud can occur instantaneously; therefore, the monitoring platform should catch them in real-time before the fraudsters flee the platform. By decoupling model inference and network querying phases, the authors aim to reduce the online inference time of a model. Specifically, they propose the Asynchronous Propagation Attention Network (APAN), which provides both synchronous inference and asynchronous propagation links. In the asynchronous link, when an interaction is created, the interaction's information (e.g., event detail and history embedding) is emailed to the k-hop neighbors of the nodes involved in the transaction. In

fact, each node has a mailbox of size x that stores the x most recent pieces of information it has received via k-hop neighbors. In this instance, rather than propagating and aggregating the neighbors' information during inference, the nodes already possess the information transmitted through the asynchronous link. Consequently, in the synchronous link, APAN just has to read the mailboxes of the nodes participating in the conversation and execute real-time inference when the interaction occurs. In fact, temporal node embeddings are calculated in the encoder as described above, and then an MLP decoder is utilized to complete a job such as link prediction or node classification based on the obtained embeddings.

Experimental results reveal that the APAN beats all other models in terms of inference time (8.7 times faster than TGN), while providing accuracy extremely near to that of TGN in link prediction and node classification, and in certain cases doing somewhat better than TGN.

A.2 Temporal Graph Network Embedding with Causal Anonymous Walks Representations

TGN generates node embedding from interactions by the message aggregation through temporal links. Also, Casual Anonymous Walk (CAW) is a state-of-the-art link prediction approach by learning edge representations. In [49], the authors attempt to combine the idea of TGN and CAW to improve the accuracy of the link prediction and node classification task. In one simple word, they have refined the message generator function and enriched it with more relevant information. The message generator is defined as:

$$m_i(t) = (s_i(t^-)||s_j(t^-)||\Delta t||e_{ij}(t)||NEF_{ij}(t)) \tag{A.1}$$

As it is clear, the only difference in the message generator function is $NEF_{ij}(t)$, which is the core contribution of the authors to TGN. $NEF_{ij}(t)$ is a trainable module that generates highly informative feature representations (link feature vector) of an link ij for any given time. This module is proposed based on CAW; however, I will not go through details about it as it is beyond the scope of this thesis. Instead, interested readers may refer to [49]. It is worth noting that other elements of their framework except the embedding generator are similar to the original TGN — $NEF_{ij}(t)$ is also added to the embedding generator.

The proposed model was tested against several baseline models such as TGN, DyRep[48], and Jodie[23]. The experimental results show that it consistently outperforms almost all models except in some cases in both transductive and inductive settings. However, it is good to mention that, in some cases, the improvement is negligible compared to the original TGN algorithm— a slight increase in the third digit.

A.3 Adaptive Data Augmentation on Temporal Graphs

In [50], the authors claim that TGNs may suffer from over-fitting and the lack of model generalization due to their complexity. Data Augmentation has been found effective in overcoming overfitting. Accordingly, the authors suggest an adaptive Data Augmentation (DA) approach called Memory Tower Augmentation (MeTA), stacking a few levels of memory modules as a tower with weight sharing. In an adaptive DA, the higher DA magnitudes are applied to the less informative parts of the input features. In MeTA, the

informativeness of links to a target node is defined in terms of both time and topology — the links closer to the target node in time and topology (e.g., direct link preferred) carry more critical information. The basic idea in MeTA is to generate a few graphs with different DA magnitudes such that the lower memory module in MeTA processes the graph of lower DA, and the higher module processes the graph of higher DA. Using these modules, MeTA accomplishes the message passing between these graphs to provide adaptively augmented inputs for predictions. It is worth noting that MeTA possesses two message-passing mechanisms to make adaptative DA feasible, such as Cross-level Propagation (propagating nodes' features across the levels) and Memory Transition (transmitting memory state of the nodes from higher to lower levels periodically).

MeTA is a general module that can be used for training DyRep [48], TGNs, etc. The experimental results indicate the effectiveness improvements of both DyRep and TGN in link prediction and node classification tasks without imposing computational overhead. TGN with MeTA was always the champion with a relatively high margin in their experiments.