Temporal Point Processes for Forecasting Events in Higher-Order Networks

A THESIS

SUBMITTED FOR THE DEGREE OF

Doctor of Philosophy

IN THE

Faculty of Engineering

BY

Tony Gracious

Computer Science and Automation
Indian Institute of Science
Bangalore – 560 012 (INDIA)

November, 2023

Declaration of Originality

I, **Tony Gracious**, with SR No. **04-04-00-10-12-17-1-14785** hereby declare that the material presented in the thesis titled

Temporal Point Processes for Forecasting Events in Higher-Order Networks

represents original work carried out by me in the **Department of Computer Science and Automation** at **Indian Institute of Science** during the years **2017–2023**.
With my signature, I certify that:

- I have not manipulated any of the data or results.
- I have not committed any plagiarism of intellectual property. I have clearly indicated and referenced the contributions of others.
- I have explicitly acknowledged all collaborative research and discussions.
- I have understood that any false claim will result in severe disciplinary action.
- I have understood that the work may be screened for any form of academic misconduct.

Date: Student Signature

In my capacity as supervisor of the above-mentioned work, I certify that the above statements are true to the best of my knowledge, and I have carried out due diligence to ensure the originality of the report.

Advisor Name: Advisor Signature

DEDICATED TO

My Parents

Acknowledgements

I am immensely grateful for the invaluable experiences I have had during my time at IISc, which have made these years the most fruitful period of my life. This institution has not only allowed me to refine my skills and learn from my mistakes but has also instilled in me the confidence and technical expertise to tackle challenging problems and compete with the best in the world.

I extend my deepest gratitude to my research supervisor, Prof. Ambedkar Dukkipati, whose significant contributions have been instrumental in helping me achieve my goals. Despite my initial status as a beginner in the field, he graciously accepted me into his research group. His unwavering support and guidance have empowered me to become self-reliant in my research endeavors. While we encountered disagreements along the way, we consistently resolved them in a mutually respectful manner. I am truly privileged to have been a part of his lab.

I would also like to sincerely thank all the professors who taught me during my master's and doctoral studies. Their instruction and guidance have given me a strong mathematical foundation, enabling me to pursue my aspirations. Additionally, I extend my thanks to Prof. Phaneendra K. Yalavarthy, Prof. Aditya Gopalan, Prof. Chiranjib Bhattacharyya, and Prof. Narasimha Murty, who served as my comprehensive viva panel and offered valuable insights by highlighting my areas for improvement.

My heartfelt gratitude goes out to the members of the SML lab, particularly my collaborators Shubham Gupta and Arman Gupta. Working alongside you has been an invaluable learning experience, and I am grateful for the knowledge and skills I have gained through our collaboration.

I would also like to acknowledge the CSA department staff for their assistance with bureaucratic procedures at IISc and for creating a conducive work environment that I could rely on daily.

Lastly, I wish to express my deepest appreciation to my parents and brother for their unwavering support and encouragement throughout this journey.

Abstract

Real-world systems consisting of interacting entities can be effectively represented as time-evolving networks or graphs, where the entities are depicted as nodes, and the interactions between them are represented as instantaneous edges. Modeling the evolution of these systems and forecasting interaction events are important for many fields, such as e-commerce, financial markets, neuroscience, etc. This is achieved using the Temporal Point Process (TPP) framework, a stochastic process that models these interactions as discrete events occurring in continuous time. The existing works on interaction forecasting are applicable only to pair-wise edges.

However, real-world interactions are much more complex than pair-wise interactions. It involves a group of entities interacting in a complex way rather than just two entities. This leads to the formation of time-evolving higher-order networks. There has not been much research to develop machine learning algorithms for event prediction in these types of networks. This thesis addresses this by providing solutions to the following problems: (i) How can we train models on the events observed in a higher-order network? (ii) Considering the number of possible events grows exponentially when problem setting changes from pair-wise to higher-order, can we forecast the next event in a scalable way? (iii) How to model the influence of interactions on nodes in a higher-order network, and how to improve its scalability? (iv) How can we incorporate relations and group structure within an interaction to an event forecasting model?

The first contribution of this thesis is a model for forecasting interactions among a group of entities as instantaneous hyperedge events in a network. In this model, we introduce a TPP on each hyperedge, with the conditional intensity parameterized by a hyperedge link predictor that uses node embeddings. We employ a dynamic node embedding strategy to account for the temporal evolution of entities with each interaction. Here, all the model parameters are learned by a mini-batch based negative log-likelihood calculation with negative sampling for incorporating unseen hyperedges. Furthermore, our model has been extended to accommodate bipartite interactions, where interactions occur between two distinct groups of entities of different types. To achieve this, we introduce a bipartite hyperedge link predictor and use separate

node embedding modules for each node type.

Secondly, we propose a model to forecast directed higher-order interactions occurring between two distinct groups of entities. Unlike the previous approach that focuses on representation learning from higher-order network events, here we also introduce a strategy to forecast hyperedges in a scalable way. For that, we employ a multi-task framework for forecasting candidate hyperedges. This involves a TPP-based model to predict the time of events on each node, followed by pairwise neighborhood and hyperedge size prediction modules for generating candidate hyperedges. This will reduce the exponential search in forecasting future hyperedges in the previous models. Then a directed hyperedge link predictor is used to identify the true hyperedge from false ones. Further, we devise a dynamic node embedding architecture that processes samples in a batch using memory network and temporal graph attention modules. This improves the scalability of the model when dealing with datasets containing a large number of events.

In our final contribution, we extend the existing interaction forecasting approaches based on TPP to accommodate real-world interactions that involve internal group structures of different types. Here, each group is associated with a specific relation type, and to address this complexity, we introduce the concept of multi-relational recursive hyperedge formation events. In this framework, hyperedges can serve as nodes within other hyperedges, creating a hierarchical structure. Further, we also introduce a contrastive learning strategy to learn model parameters without using the intractable likelihood of TPP.

In conclusion, this thesis emphasizes and demonstrates the importance of employing higher-order temporal network models to forecast interactions in real-world systems accurately. To achieve this, we created deep neural network based approaches for dynamic node embedding to extract information from historical data and link predictors to model the occurrence of the edge formation event. Then devised different strategies to train these models and forecast events from them.

Contents

List of Figures

List of Tables

List of notations and abbreviations

Terminology related to Temporal Point Process (TPP).

Notation	**Description**
t	time
T	End time
n	Total number of events
d	Embedding size
$t_i \in [0, T]$	ith event time
$\mathbb{N}(t)$	Counting process, number of events happened till time t
$\mathcal{C}$	Set of categorical items $\mathcal{C} = \{c_1, .., c_n\}$
$\Delta t_i = t_i - t_{i-1}$	the ith inter-event time, assuming $t_0 = 0$
$\mathcal{E}(t)$	A realization of a marked TPP $\{(t_1, e_1), \ldots, (t_i, e_i)\}$ till $t_i \leq t$
$\lambda(t)$	Conditional intensity function at time t
$\Lambda(t)$	Compensator function at time t
$\mathrm{d}\tau$	Infinitesimal change in τ
$\mathrm{P}(t\|\mathcal{E}(t_i))$	Probability density function for $i+1$th event
$\mathrm{F}(t\|\mathcal{E}(t_i))$	Cumulative density function for $i+1$th event
$\mathrm{S}(t\|\mathcal{E}(t_i))$	Survival function for the $i+1$th event
$\lambda_m(t)$	Conditional intensity function corresponding to mark m
$\mathrm{P}(\mathcal{E}(t))$	Probability density of the TPP
$\mathbf{h}(t)$	Hidden state at time t
$\mathcal{L}$	Likelihood
$\mathcal{LL}$	Log likelihood
$\mathcal{NLL}$	Negative log likelihood

Abbreviations.

Notation	Description
TPP	Temporal Point Process
PDF	Probability Density Function
CDF	Cumulative Distribution Function
SF	Survival Function
MCMC	Markov Chain Monte-Carlo
MC	Monte-Carlo
KG	Knowledge Graph.
TKG	Temporal Knowledge Graph.
GRU	Gated Recurrent Unit
RNN	Recurrent Neural Network
LSTM	Long Short-Term Memory
GNN	Graph Neural Network

Standard quantities and functions.

Notation	Description
$\mathbb{I}_{(\cdot)}$	Indicator function. $\mathbb{I}_p$ returns one if condition p is true, else zero
$\ln(\cdot)$	Natural logarithm
$\mathsf{O}(.)$	Order of complexity
$\mathsf{C}(.)$	All the combinations of a set of items
$\mathbb{N}$	Set of natural numbers, $\{1, 2, 3, \dots\}$
$\mathbb{R}$	Set of real numbers
$\mathbb{R}^+$	Set of positive real numbers
$\mathcal{X} \backslash \mathcal{Y}$	Elements of set $\mathcal{X}$ that do not belong to set $\mathcal{Y}$
$\lvert\mathcal{X}\rvert$	Cardinality of set $\mathcal{X}$
$\mathbf{I}$	Identity matrix
$\mathbf{1}$	A vector whose each element is equal to one
$\lvert\lvert \cdot \rvert\rvert_2$	Euclidean norm of a vector
$\mathrm{E}[\cdot]$	Expectation with respect to a given distribution
$\mathrm{P}(\cdot)$	Probability density function
$\sigma(.)$	Sigmoid function with $\sigma(x) = \frac{1}{1+\exp -x}$

Terminology related to networks.

Notation	**Description**
$\mathcal{V}$	Set of nodes or vertices, $\mathcal{V} = \{v_1, v_2, \ldots, v_{\lvert\mathcal{V}\rvert}\}$.
h	Hyperedge, $h \subseteq \mathcal{V}$
$\mathcal{H}$	Set of hyperedges
k	Hyperedges size, $k = \lvert h \rvert$
$\mathcal{G}$	Hypergraph
$\mathcal{G}(t)$	Temporal Hypergraph
$\mathbf{A}$	Adjacency matrix of the network
$\mathbf{A}(t)$	Adjacency matrix of a dynamic network at time t
$\lambda_h(t)$	Conditional intensity function of interaction h

Chapter 1

Introduction

Interactions among entities often drive real-world systems. These interactions significantly impact the behavior and properties of the entities, which, in turn, affects the future evolution of the system. Social networks serve as a prime example of such systems, where individuals interact and influence one another, leading to the formation and dissolution of connections and user participation changes. Similar dynamics can be observed in biological systems, economic markets, neuroscience, and other domains. Building AI systems that have the ability to forecast these interactions based on historical data is a crucial problem. This involves developing models that can track the temporal evolution of entities by learning meaningful representations from historical data. These models can provide solutions to downstream tasks such as link prediction, classification, or community detection.

Figure 1.1a shows interactions between geopolitical actors during the prelude of the Russian invasion of Ukraine. Analyzing and forecasting the impact of these interactions on trade and geopolitical relations could hold significant potential in areas like policymaking and financial markets. For instance, the invasion led to disruptions in the supply chain resulting in global food shortage (Ben Hassen and El Bilali, 2022). By anticipating such developments earlier, policymakers could formulate effective mitigation plans to address similar challenges in the future. Further, Figure 1.1b presents an instance of interaction between users on QA websites like Stack Overflow and Quora. In this scenario, a user named "Alice" poses a question, and other participants engage in activities such as editing and offering answers to the query. Developing models to monitor these interactions can identify users who tend to respond promptly and correctly to questions. It also helps to determine the optimal time to post queries to maximize their visibility. Additionally, these types of models can be built for deployment on social media platforms like Twitter or Facebook, enabling efficient product advertisement and garnering maximum exposure for the promoted items.

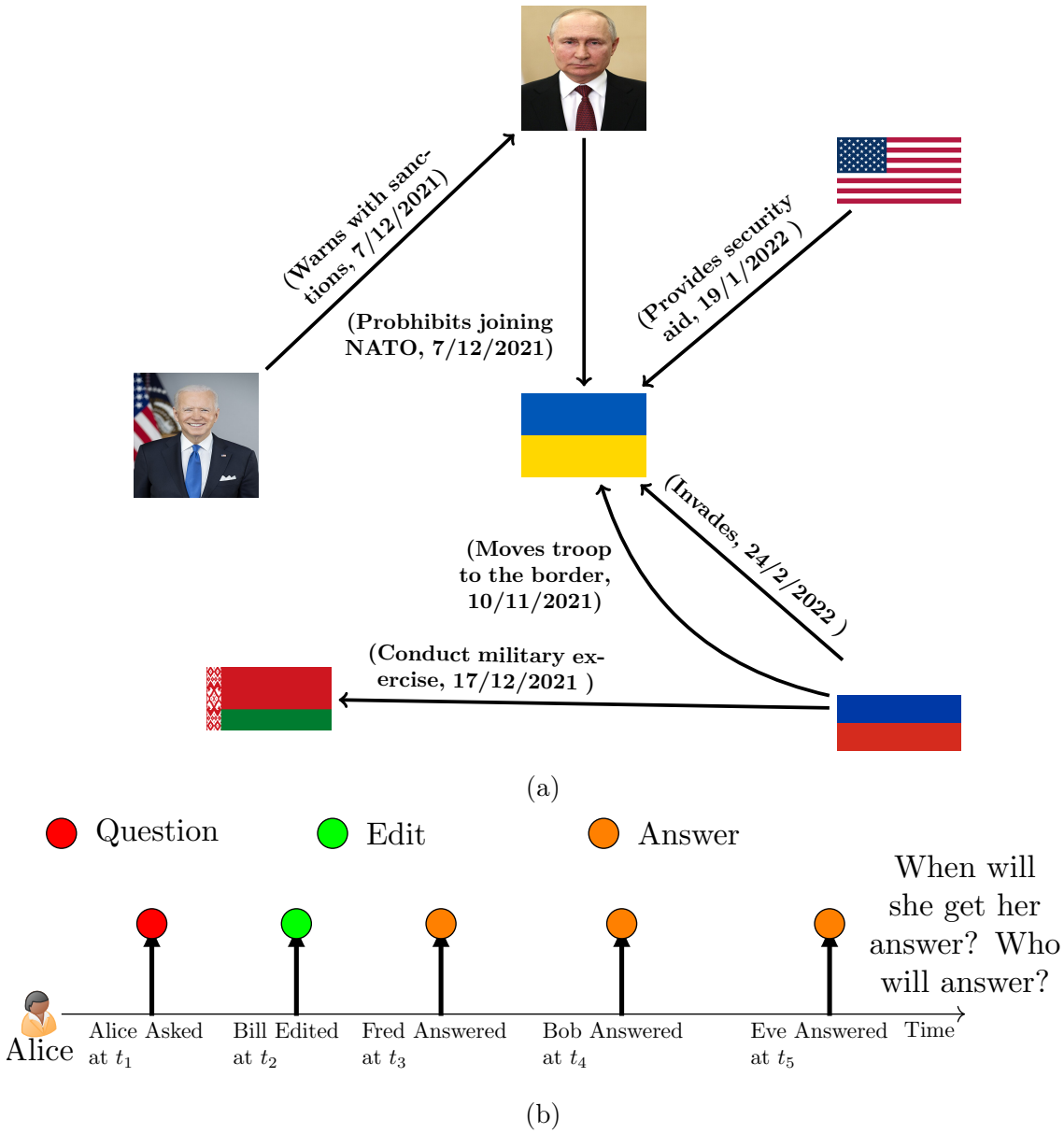


Figure 1.1: Figures demonstrate examples of interactions occurring between entities in the real world.

Traditional approaches for forecasting have primarily focused on time series analysis, where sequences of features are associated with a particular phenomenon, and the goal is to predict the future based on historical data. Popular auto-regressive models like ARMA, ARIMA, and SARIMA (Box and Jenkins, 1990) have been commonly used for this purpose. However, these models have limitations in terms of their expressive power. In the era of deep learning by using powerful function approximators like Recurrent Neural Networks (Hochreiter and Schmidhuber,

1997) and Transformers (Vaswani et al., 2017), we can capture complex relationships and achieve state-of-the-art performance in forecasting tasks. Nevertheless, these models assume regular intervals of observations, which is unrealistic given that real-world interactions often occur at irregular intervals. For instance, website visits by a user can vary widely in timing, frequency, and duration. These interactions are similar to discrete events happening in continuous time and present a challenge for traditional time-series forecasting techniques that rely on uniform time intervals. Figure 1.2 illustrates interaction as events happening in real-time, and there are two distinct types of interactions shown by the type of the event. Here, to apply time-series analysis based models, the observation period is discretized into equal duration bins, and each bin is labeled according to the events occurring within that timeframe. However, doing a time-series analysis involves the following challenges. Firstly, determining the appropriate bin length requires some domain knowledge. Secondly, the aggregation of events becomes problematic since multiple event types can occur within a single bin. Additionally, the conventional time-series models cannot answer temporal queries, such as forecasting the time of occurrence of an event.

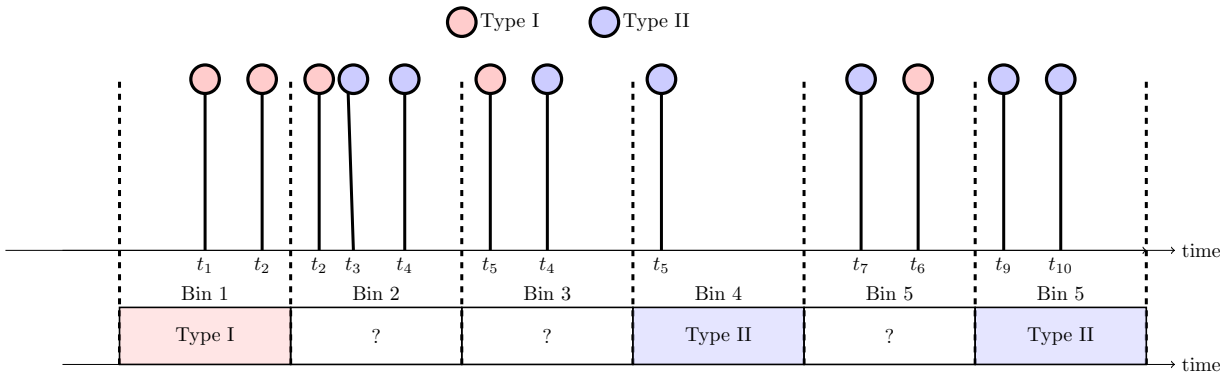


Figure 1.2: The figure shows interactions as real-time events and the challenges involved in time-series analysis on interaction forecasting. Here, each event is associated with type, and two types of events are happening.

The theory of the Temporal Point Process (TPP) has been developed to model events occurring at irregular time intervals (Daley and Vere-Jones, 2003). Here, each event is associated with two features: its time of occurrence and a mark representing additional information about the event, such as its type/class or a feature vector associated with that event. For example, in earthquake modeling, an event is described by its occurrence time and the intensity of the earthquake, which is the mark associated with that event. In a web traffic modeling system, each event corresponds to a user request, characterized by its arrival time and the type of request made. For modeling these, one can define a probability distribution over the time at which the

event will be observed, and the mark of the event is modeled with a suitable distribution based on its nature while conditioned on the time of the event.

Early works involve using a non-homogeneous Poisson process with intensity as a function of time for modeling telephone traffic data and insurance claims (Cramér, 1969). However, these works assume events are independent of each other. This assumption was relaxed by developing the theory of Hawkes Process, which uses kernel functions to model the influence of past events on the future (Hawkes, 1971). This is also known as a self-exciting process, where the likelihood of future event occurrence increases briefly with each instance of an event. Further, algorithms are developed for simulating events from this nonhomogeneous Poisson processes (Lewis and Shedler, 1979, Ogata, 1981). However, these works use prior information about the form of intensity function that may not accurately reflect actual data. The non-parametric Hawkes process models are developed to learn a flexible function for the intensity of the events in a data-driven manner (Lewis and Mohler, 2011).

Only recently has the machine learning community started adapting TPP based models in developing algorithmic solutions for event prediction problems. These models are developed to improve the representation capabilities of the statistical TPP models mentioned in the early paragraph. Earlier works in this direction use a continuous time recurrent neural network model for learning hidden state of the process, which is then used to parameterize the conditional intensity function of the TPP (Du et al., 2016, Mei and Eisner, 2017). However, these models cannot model long-range dependencies and hence cannot model long sequences. So, recent works have started using transformer based architecture to model the hidden state of the TPP (Zuo et al., 2020, Zhang et al., 2020, Lin et al., 2022). All these models are grouped under the category of Neural TPP as they use neural networks to parameterize the probability distributions of the TPP.

Despite the successful applications of these models in diverse domains, including seismology (Ogata, 1998), financial markets (Filimonov and Sornette, 2012), and social sciences (Lewis et al., 2011), they encounter difficulties when dealing with real-world interactions involving a large number of entities. The complexity arises from the existence of numerous ways to group entities, resulting in various types or classes of marks associated with the events. A naive application of Temporal Point Process (TPP) models, without accounting for the correlations between these marks, gives rise to a proliferation of parameters, as distinct parameters are required for each event type. Consequently, such an approach suffers from poor generalization performance in unseen data during testing. Moreover, these models struggle to capture interactions not observed in the training data, analogous to the problem of handling unobserved classes in classification tasks. Graph models offer a solution by representing entities as nodes

and using edges to represent interactions. Since real-world interactions are not limited to two entities, higher-order networks are a more accurate approximation of them. A hypergraph is an example of this type of network, as its edges (hyperedges) involve a variable number of nodes. By leveraging network representation and embedding learning techniques, TPP models can be effectively parameterized to capture the dynamic nature of these processes. Next, we delve into related research in this direction, shedding light on the need for event forecasting on higher-order networks.

Event prediction on Networks. TPPs have been developed for predicting events on time evolving networks. Here, the model considers instantaneous pairwise edges as an event. These works can be classified into statistical and deep learning based models, and the models are trained by minimizing the negative log-likelihood of TPP on the training data.

Models in the first category primarily employ the Hawkes process, a special form of TPP, to model interactions between a pair of entities. These use node embeddings to parameterize the kernel and baseline intensity functions. These properties can be the latent community structure (DuBois et al., 2013, Matias et al., 2018, Junuthula et al., 2019, Arastuie et al., 2020, Soliman et al., 2022) or the latent features of the nodes in the edge (Yang et al., 2017, Huang et al., 2022, Passino and Heard, 2022). Here, the kernels mostly try to model the properties such as homophily, reciprocity, and self-excitation of the pair-wise interactions in the network. While these models perform well in some domains, they often rest on strong assumptions that may not always hold in practice. For instance, these models generally assume that the impact of events on future outcomes is always positive, which does not account for situations where negative interactions may inhibit future events. In social networks, for example, negative interactions between individuals can decrease social activity and lower the chances of future events occurring. Additionally, these models generally assume that the influence of events is additive, whereas, in reality, events can have complex and non-additive effects. For instance, in financial markets, the interaction between market forces can create non-linear and unpredictable impacts on the occurrence of future events. Finally, these models typically assume that events have an instantaneous effect on the process, while in reality, events can have delayed effects, such as the incubation period of an epidemic.

The deep learning based models leverage temporal graph representation learning to generate continuous time node embeddings that are then used to parameterize the conditional intensity function of the TPP. These techniques have been used in recommendation systems where continuous-time user and item embeddings are used for parameterizing TPP to forecast items of interest to a particular user. These models assume that entities are of two kinds, users and items, and all interactions are between them. DeepCoevolve (Dai et al., 2016) uses a re-

current neural network to generate continuous node embeddings, and interactions are modeled as Rayleigh process (Ghosh, 2009), a form of TPP. DSPP uses a combination of discrete-time models with a time projection layer to learn continuous time node embeddings, and the intensity function is parameterized by the inner product score of user and item embeddings (Cao et al., 2021). Then there are studies for modeling user interactions in social media and the real world. Here, they assume that all entities are homogeneous or of the same kind, and interactions are between them. DyRep uses graph attention features and recurrent neural networks for continuous node embedding (Trivedi et al., 2019). Then the intensity function is parameterized using the bilinear product of embeddings of interacting entities. Similarly, GNPP (Xia et al., 2022) uses graph based message passing framework with Fourier time features (da Xu et al., 2020) to find the embeddings of the edge that is then used to parameterize the intensity function.

Link Prediction in Temporal Graphs. Link prediction in temporal graphs deals with events as a binary classification problem on edges at the time of interest. They can be achieved by learning a dynamic node representation trained to forecast future links using historical interaction data. These studies can be categorized based on how they model time evolution: (i) discrete-time models and (ii) continuous-time models.

In discrete time models, time is divided into equal-sized bins, and interactions are aggregated within each bin to create a sequence of graph snapshots. Earlier works, such as Sarkar and Moore (2005), Zhu et al. (2016), Goyal et al. (2018) and Zhou et al. (2018), learn separate node embeddings for each graph snapshot by training on link prediction loss with temporal smoothing techniques to capture dependencies across different snapshots. However, these approaches have limitations regarding their expressive power, as they primarily incorporate temporal information through regularization techniques without explicitly modeling the dynamics of the network over time. More recent models have utilized techniques such as recurrent neural networks (Hochreiter and Schmidhuber, 1997) and attention mechanisms (Vaswani et al., 2017) to capture the temporal evolution of graphs. For example, DynAERNN uses a recurrent model to capture the temporal evolution of node embeddings and is trained to reconstruct the adjacency matrix (Goyal et al., 2020). DySAT employs graph attention to capture structural information and temporal self-attention to incorporate temporal information into the node embeddings (Sankar et al., 2020). NLSM is a probabilistic model that uses a dynamic latent space model for node embeddings and is trained to reconstruct adjacency matrix using variational inference with a recurrent neural network parameterizing the variational distribution (Gracious et al., 2021). Despite their effectiveness, there are limitations to using discrete-time models to represent real-world systems, as discretization results in information loss, and the choice of bin size requires domain knowledge.

There has been a huge surge in using continuous time models for temporal graphs in recent years as it is a more accurate representation of real-world systems. This is done by defining a scoring function that takes continuous time embeddings of a node as input, and training is done to give higher scores for the events observed at that time. JODIE uses RNN to update the node embeddings based on the pair-wise interaction features and time embedding projection layer to model evolution of node during inter-event period (Kumar et al., 2019). TGAT uses graph attention network that uses Fourier time features to aggregate historical interactions to generate temporal node embeddings (da Xu et al., 2020). TGN is an extension of TGAT by incorporating a recurrent neural network based node embedding update stage (Rossi et al., 2020). However, all these methods are trained on supervised link prediction loss and can only infer the presence and absence of an event type at a particular time. At the same time, TPP based methods can also predict the time of occurrence of an event type.

Temporal Knowledge Graphs. Knowledge Graphs (KGs) provide a data structure for storing facts in the form of triplet `(s, p, o)` as the edges of a multi-relational graph. Here, `s` and `o` denote the subject and object entities, respectively, and `p` is the relation between them. An example of KG data is `(Amazon River, LocatedIn, South America)`, here representing the location of river `Amazon River`. Major companies like Google, IBM, and Microsoft build and use these for different types of information retrieval tasks, such as improving search results, chatbots, and virtual assistants. However, these graphs are far from complete, and there is a lot of missing information that could be inferred from the topology of these graphs. This has been done by learning embeddings or representations for entities and relations in the KG by training a scoring function to predict the presence of an edge. Since only true samples are observed, the negative samples are created by corrupting true samples by randomly changing the subject, relation, or object. These scoring functions can then be categorized into distance and similarity based methods. TransE (Bordes et al., 2013), TransH (Wang et al., 2014) and RotatE (Sun et al., 2019) are some of the popular distance based methods which learn by trying to minimize the distance between subject and objection embeddings of the true samples using a relation based transformation. DisMult (Yang et al., 2015), ComplEx (Trouillon et al., 2016) and SimplE (Kazemi and Poole, 2018) are popular similarity based scoring functions that try to increase the similarity of true samples in a relations space. A more detailed survey with the pros and cons of these techniques can be found in this survey done by Wang et al. (2017). Recently multi-relational graph neural networks techniques have been used for KG completion to achieve state-of-the-art performance using the graph neighborhood based feature extraction (Schlichtkrull et al., 2018, Vashishth et al., 2020, Teru et al., 2020).

However, real-world entities evolve with time, especially entities involved in political events.

The edge `(Donald Trump, President of, USA)` is only valid from January 20, 2017, to January 20, 2021. One can use the quadruplet form of `(s, p, o, t)` to represent these interactions, with `t` being the timestamp at which it is valid. Knowledge Graphs (KGs) that incorporate such temporal information are referred to as Temporal KGs (TKGs). Research in TKG can be categorized into two kinds: i) interpolation and ii) extrapolation. Given a TKG consisting of interactions valid during the time from t_0 to t_n, the interpolation task aims to find the missing interactions that happened during that interval. Early works in this direction augment the static link prediction with time based transformation. This involves Garcia-Duran et al. (2018), which uses an LSTM to encode the relation and time into modified relation vectors and train on the static link prediction methods DisMult and TransE. Another approach is to divide graphs into discrete snapshots and then project the embeddings of subject, object, and relation to time-based hyperplane, and learning is done based on TranSE scoring function (Dasgupta et al., 2018). Recent works on TKG focus on the extrapolation task of forecasting interactions at $t > t_n$, which is more challenging. The literature has approached this task with dynamic representation learning frameworks trained on TPP loss or link prediction loss. Know-Evolve (Trivedi et al., 2017) uses a Rayleigh process (Ghosh, 2009) with its intensity parameterized by dynamic embedding of subject, object, and relation embeddings. GHNN aggregates the historical interaction of an entity using graph neural networks to parameterize the intensity function of TPP (Han et al., 2020). On the other hand, RE-NET (Jin et al., 2020) and Evolve-KG (Park et al., 2022) use graph neural network based auto-regressive model to forecast KG links in a discrete-time model.

Events in Higher-Order Networks. While forecasting events on pairwise networks has been extensively studied, predicting events on higher-order networks involving edges with more than two entities is still a relatively novel and emerging area of research. These types of models can be applied in areas such as recommendation systems in e-commerce websites where users and products interact. In these systems, users often purchase multiple items simultaneously, and using pairwise models to forecast events will ignore the correlation between items. For example, there might be products that may not be bought together, like a person might buy `(bread, butter)` or `(bread, jam)` but may never buy `(bread, butter, jam)`. Creating effective recommendations that address such complex interactions can substantially enhance user subscriptions and overall satisfaction. Likewise, these interactions are also evident in QA websites, where users collaborate to answer questions. Identifying a group of domain experts who consistently provide valuable responses can lead to increased user engagement and an improved user experience. This will promote meaningful user collaboration on such platforms and elevate the quality of information shared. Similarly, we can observe higher-order network

events in communication systems, such as email exchanges with a sender and multiple receivers, and also in citation networks where multiple authors collaborate to publish an article. Hence, creating models that forecast these events will have immense application in diverse fields.

Existing studies have primarily focused on forecasting specific aspects of the events on higher-order networks for analysis purposes. Recent work by Benson et al. (2018) introduced the concept of simplicial closure, extending the triadic closure property of pairwise networks to higher-order networks. This property predicts that a group of interacting entities will coexist in a single interaction. This work demonstrated this property using a classification model with network analysis based features, determining whether three nodes would coappear in an interaction based on their past interaction patterns. Building upon this, Liu et al. (2022) extended the simplicial closure property to a deep learning model that forecasts the type and timing of interactions between three nodes based on their historical information. Further, in Benson et al. (2018), a set based model is used to find the future interactions as a union of historical sets. In addition to these, there are works that model the statistical properties of events on higher-order networks like the evolution of ego networks (Comrie and Kleinberg, 2021), persistence (Choo and Shin, 2022), reciprocity (Kim et al., 2022), and motifs (Lee and Shin, 2023b). However, these works do not model the entire dynamics of the interactions by considering the evolution of the entities involved.

This thesis explores methods for forecasting events on higher-order networks using TPP. The primary challenge in achieving this is the computationally expensive likelihood calculation. This is because the number of possible event types is exponential in the order of the number of entities, as each entity combination is represented as a distinct event type, and most of the event types are not observed. Another challenging aspect involves computing the gradient of the TPP likelihood during training using modern automatic differentiation frameworks. The requirement of loading the entire data into memory for gradient back-propagation becomes impractical, particularly in real-world systems with a substantial number of events. Hence, to create a TPP model, one needs a training strategy to approximate the likelihood computationally efficiently while working with mini-batches created by dividing the observation period. Further, forecasting future events also becomes challenging as it involves searching through all the possible event types. Hence, the model also requires a forecasting strategy that limits the possible candidate to a small subset of all possible types of events to infer predictions from these models.

Another challenge is in designing neural architectures that can model the dynamics of interacting entities and forecast the events. This involves proposing a dynamic node embedding strategy that aggregates the historical interaction information and a link predictor to model

the occurrence of the interaction. Here, the dynamic node embeddings are to be updated when the respective nodes are involved in an event based on the interaction features as entities evolve with each interaction. In pairwise model, the node embedding is updated based on the embedding of the other node in the interaction. However, higher-order networks involve multiple nodes in a single event. Moreover, updating node embeddings for each event is a sequential operation. It will create scalability issues when training on a large number of events. This is due to the inability to perform batch processing of samples, as each sample depends on the entire previous history.

Further, the link prediction module must predict interactions involving a variable number of entities. Most of the research on higher-order networks are on hyperedge link prediction, which does not model internal groups and relations involved in an interaction. For example, an email exchange involves a sender, recipients, and carbon-copied recipients (cced) addresses. Similarly, in citation networks, the authors of a scientific article are ordered according to their contributions to the work. Here, the author's position is the grouping, and multiple authors can be in the same position.

1.1 Contributions

As explained in the previous section, earlier works focused on forecasting events occurring in pairwise networks. Ours is the first work that explores the problem of modeling edge formation events in a higher-order network. To achieve this, the model needs a dynamic node embedding architecture to aggregate history into continuous time representations of the nodes and a link prediction to model the probability of the occurrence of the event while considering the complex relationship between the entities involved. The parameters of the model are to be trained using a strategy that approximates the likelihood calculation while also working on mini-batches. Through this thesis, we devise different ways to address these challenges by asking the following questions.

Question 1: *How can we train models on the events observed in a higher-order network?* In this thesis, we have devised three methods to train models from events in a higher-order network.

In Chapter 3, we propose a model for forecasting hyperedge formation events. Here, we focus on homogeneous interaction between a group of entities. Our model uses dynamic node embeddings to parameterize the conditional intensity function based on the hyperedge link predictor. In this, we address the challenge of calculating likelihood, which is not analytically feasible. This is achieved using a mini-batch training procedure by dividing the training dura-

tion into batches, and likelihood is calculated within these batches. Here, unseen event types are handled through negative sampling.

We follow a similar approach in Chapter 5 by using the conditional intensity function formulation of TPP. However, instead of calculating the likelihood, we create a noise-contrastive training strategy by simulating a noise distribution, and model parameters are trained to discriminate samples generated from the true distribution, which is observed data, from noise-generated samples. Here, noise distribution is created in a way to incorporate unseen event types.

In Chapter 4, we use a temporal link prediction-based loss for forecasting events in a higher-order network as edge formation. Given the time of the event, we trained a classifier to predict the true edge from the candidate edges generated through a TPP module.

Question 2: *Considering the number of possible events grows exponentially when problem setting changes from pair-wise to higher-order, can we forecast the next event in a scalable way?*

In Chapter 4, we address this issue using a generator model based on temporal point processes. This model forecasts a subset of event types that are likely to be candidates for the next event. Thereby filtering out the vast number of unseen events as probability zero events. The forecasting task is divided into three modules: (1) a temporal point process module to identify the nodes participating in the upcoming event, (2) a module to forecast the adjacency matrix and sizes of interactions involving these nodes, and (3) a hyperedge link prediction module to filter out the actual events from a set of candidate events generated from the previous stage. Here, all modules utilize the dynamic embeddings of nodes as input for forecasting.

Question 3: *How to model the influence of interactions on nodes in a higher-order network, and how to improve its scalability?*

The models in Chapter 3 and 4 use a sequential update of node embeddings with each event based on the interaction features. Since each event involves a variable number of nodes, we use an attention-based aggregator to compute the interaction features.

However, sequentially updating node embeddings do not allow batch processing of data, and it will affect models' scalability to apply on networks with a large number of events. In Chapter 4, we use dynamic node embeddings architecture that stores features of interactions in batch and is used to update the node embeddings before the start of the next batch. To avoid embedding staleness problems, a temporal graph attention architecture is used to extract information from its previously interacted nodes.

Question 4: *How can we incorporate relations and group structure within an interaction to an event forecasting model?*

In Chapter 3, we extended the model to bipartite hyperedge events to model interactions occurring between two groups of entities of different types. These kinds of interactions are common in recommendation systems where nodes representing users interact with nodes representing products. We model these using a node type-specific encoder module for dynamic node embeddings and a bipartite hyperedge-based link predictor to parameterize the conditional intensity function of TPP.

Additionally, in Chapter 4, we created a model to forecast directed hyperedge events to model the interaction between two groups of homogeneous entities. These types of interactions can be seen in cryptocurrency transactions where there are a set of senders and another set of receivers. Here, we created a dynamic node embedding module that uses temporal graph attention to extract information from historical interactions where nodes are in different groups. Then a directed hyperedge link prediction module is used to classify the true hyperedge event from false ones.

We further extend the model from Chapter 3 to accommodate real-world interactions that involve a variable number of groups of different types. Here, each group is associated with a specific relation type. Examples of these types of interaction can be seen in email exchanges, where there are sender, recipients, and carbon-copied recipients (cced) addresses. Here, each group can be represented as hyperedge with associated relation indicating their place in the email sent. To accommodate this, Chapter 5 introduces multi-relational recursive hyperedge formation events. In this framework, hyperedges can serve as nodes within other hyperedges, creating a hierarchical structure. Here, we introduce dynamic node embeddings and hyperedge link predictor that uses these intricate relations of the nodes in the interaction.

1.2 Prelimiaries

1.2.1 Temporal Point Process: Notations and Basic Definitions

A TPP is a continuous-time stochastic process that models discrete events in time t_i, $t_i \in \mathbb{R}^+$. It defines a Probability Density Function (PDF), $\mathrm{P}((t_{n+1}, e_{n+1})|\mathcal{E}(t_n))$, for future event (t_{n+1}, e_{n+1}) by observing historical events till time t_n, $\mathcal{E}(t_n) = \{(t_1, e_1), (t_2, e_2), \ldots, (t_n, e_n)\}$. Here, t_{n+1} is the time of the event, and e_{n+1} is the type of the event. Given an observation period of $[0, T]$ and N observations. The TPP has the following likelihood,

$$\sum_{n=1}^{N} \log \mathrm{P}((t_n, e_n)|\mathcal{E}(t_{n-1})) + \log \mathrm{P}(t_{N+1} > T|\mathcal{E}(t_N)).$$

1.2.2 Hypergraphs: Notations and Basic Definitions

Hypergraph. A hypergraph $\mathcal{G} = (\mathcal{V}, \mathcal{H})$ is a generalization of a graph when edges can be formed with a variable number of nodes. Here $\mathcal{V} = \{v_1, \ldots, v_{|\mathcal{V}|}\}$ is the set of nodes, $\mathcal{H} = \{h_1, \ldots, h_{|\mathcal{H}|}\}$ is the set of hyperedges/hyperlinks in the network, and $h_i \subset \mathcal{V} \, \forall \, h_i \in \mathcal{H}$. The size/cardinality of a hyperedge $|h_i|$ is denoted by the number of nodes contained in it and is denoted by k. In this, if all the sizes of hyperedges are two, $|h_i| = 2$, we will get the regular pairwise network.

Directed Hypergraph. A directed hypergraph $\mathcal{G} = (\mathcal{V}, \mathcal{H})$ is a generalization of a directed graph when there is a directed relation between two sets of entities, $h = (h^r, h^l)$. Here, $h^r \subset \mathcal{V}$ is the right hyperedge, and $h^l \subset \mathcal{V}$ is the left hyperedge. The size of a hyperedge h is a tuple containing the size of right and left hyperedges, $k = (k^r = |h^r|, k^l = |h^l|)$.

Bipartite Hypergraph. Unlike a directed hypergraph, in a bipartite hypergraph, hyperedges are between entities of two different types, $\mathcal{V}^r = \{v_{1^r}, \ldots, v_{|\mathcal{V}^r|}\}$, and $\mathcal{V}^l = \{v_{1^l}, \ldots, v_{|\mathcal{V}^l|}\}$. Here, $h^r \subset \mathcal{V}^r$, $h^l \subset \mathcal{V}^l$, and $\mathcal{V}^r \cap \mathcal{V}^l = \emptyset$.

Recursive Hypergraph. A Recursive Hypergraph $\mathcal{G} = (\mathcal{V}, \mathcal{H})$ is a generalization of Hypergraph, where hyperedges can contain nodes and other hyperedges. Let, $\mathcal{H}^l = \mathsf{C}(\cup_{i=0}^{l} \mathcal{H}^i)$ if $l \geq 1$, where $\mathcal{H}^0 = \mathsf{C}(\mathcal{V})$. A recursive hypergraph is of depth l if $\mathcal{H} \subset \mathcal{H}^l$.

Temporal Hypergraph. Temporal hypergraph is denoted by $\mathcal{G}(t) = (\mathcal{V}, \mathcal{H}, \mathcal{E}(t))$. Here, $\mathcal{E}(t) = \{(e_1, t_1), \ldots, (e_n, t_n), \ldots, (e_N, t_N)\}$ is the ordered set of historical events till time t with $e_n \in \mathcal{H}$ and N is number events occurred. $\mathcal{E}(t_a, t_b)$ is the ordered set of events observed during the interval $[t_a, t_b]$.

Temporal Multi-relational Recursive Hypergraphs. A Temporal Multi-relational Recursive hypergraph is denoted by $\mathcal{G}(t) = (\mathcal{V}, \mathcal{H}, \mathcal{R}, \mathcal{E}(t))$, where $\mathcal{R}$ is the set of relations, and $\mathcal{E}(t) = \{(e_1, t_1), \ldots, (e_n, t_n), \ldots, (e_N, t_N)\}$ is the ordered set of historical events till time t with $e_n \in \mathcal{H}$. In this each hyperedge $h \in \mathcal{H}$ is formed of a set of recursive hyperedges, $h = h^l = \{(h_1^{l-1}, r_1^{l-1}), \ldots, (h_i^{l-1}, r_i^{l-1}), \ldots, (h_{k^{l-1}}^{l-1}, r_{k^{l-1}}^{l-1})\}$ of depth l. Here, $r_i^{l-1} \in \mathcal{R}$ is the relation of hyperedge h_i^{l-1} with respect to hyperedge h^l, and $h_i^{l-1} \in \mathcal{H}^{l-1}$. Here, k^{l-1} is the number of relational groups in hyperedge h.

1.3 Outline

In this section, we will summarize the organization of the thesis.

Chapter 2. This chapter provides an overview of the preliminaries that are required for understanding the contribution of the thesis. We provide a detailed account of TPPs and how

they can be used for forecasting events in networks. Finally, present our work, a practical application of TPP to COVID-19 forecasting and designing lockdown policies.

Chapter 3. This chapter describes a model for forecasting interactions among a group of entities as events in a hypergraph. Here, we use TPP to model event occurrence with hyperedge as the event type. Then for parameterizing the TPP, we use a hyperedge link predictor that uses dynamic node embeddings as input. The model parameters are trained on negative log-likelihood based loss. It also contains extensive experiments to show the architecture's efficiency and superiority over pairwise models. Further, we extend this model to forecast events in bipartite hypergraphs. Here, interactions are between two groups of entities that are of different types.

Chapter 4. This chapter deals with forecasting events occurring in a directed hypergraph. Unlike the approach in the previous chapter that focuses on embedding learning from interactions, here we also introduce a strategy to forecast events in a scalable way. Here, we use a multi-task framework that involves a TPP-based model to predict the time of events on each node, followed by pairwise neighborhood and hyperedge size prediction modules for generating candidate hyperedges. This reduces the search space in forecasting hyperedges. Further, we describe a novel directed hyperedge link predictor and a scalable dynamic node representation module that uses temporal graph attention techniques.

Chapter 5. In this chapter, we extend the existing interaction forecasting approaches based on TPP to accommodate real-world interactions that involve internal group structures of different types. Here, each group is associated with a specific relation type, and to address this complexity, we introduce the concept of multi-relational recursive hyperedge formation events. In this framework, hyperedges can serve as nodes within other hyperedges, creating a hierarchical structure. We also describe a contrastive learning training procedure that avoids the intractable likelihood calculation.

Chapter 6. This is the last chapter of the thesis, where we make some concluding remarks and describe some directions for future work.

Chapter 2

Background

This chapter provides an overview of the preliminaries that are required to understand the contribution of the thesis. We provide a detailed account of TPPs and how they can be used for forecasting events in networks.

Sections 2.1 till 2.5 of this chapter focus on TPP and are based on works by Rasmussen (2018), Chen (2016), Rodriguez and Valera (2018), Shchur (2022) and Mei (2021). For Neural TPP, we aggregated content based on the surveys done by Lin et al. (2021) and Lin et al. (2022). Additionally, we have also referred to the surveys done by Yan (2019) and Shchur et al. (2021), which provide an overview of recent advancements in both statistical and deep temporal point process. In Section 2.5, we provide a brief account of recent results related to TPP on graphs. Finally, in Section 2.6, we present our work, a practical application of TPP on COVID-19 forecasting and designing lockdown policies.

2.1 Temporal Point Process

A TPP is a continuous-time stochastic process that models discrete events in time, $\{t_i\}_{i=1}^{n}$. Here, $t_n \in \mathbb{R}^+$ is the time at which nth event occurred. The time of an event is a random variable, and the PDF is defined for the future event time t_{n+1} by observing historical events till the last event time t_n, $\mathcal{E}(t_n) = \{t_i\}_{i=1}^{n}$. Alternately, one can view TPP as a counting process $(\mathbb{N}(t))$ on time t,

$$\mathbb{N}(t) = \sum_{i=1,2,\ldots} \mathbb{I}_{t_i \leq t}, \tag{2.1}$$

with $\{t_i\}_{i=1,2,\ldots}$ the times at which events have happened. Here, $\mathbb{I}_{t_i \leq t}$ is an indicator function that returns one when the argument $t_i \leq t$ is true and zero otherwise. Figure 2.1 shows an

example of TPP as a counting process. Here, the value of $\mathbb{N}(t)$ is increased by one with each occurrence of an event.

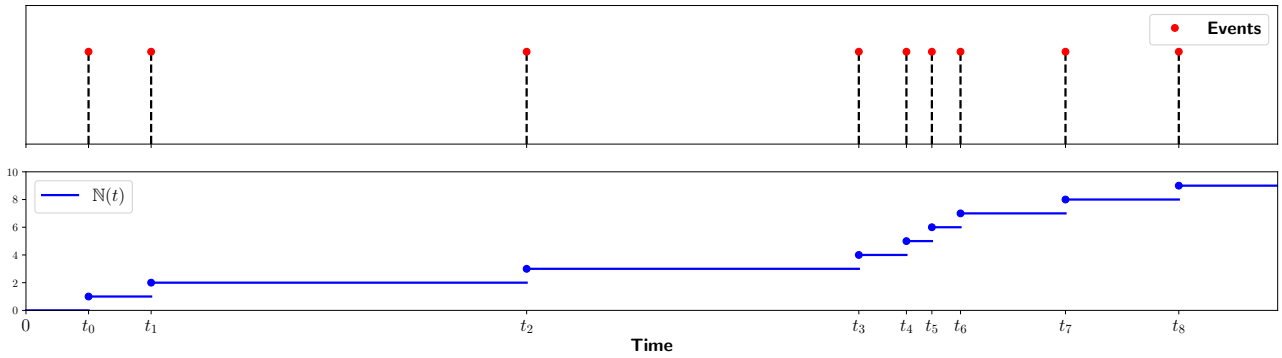


Figure 2.1: TPP events and counting process. Here, the value of $\mathbb{N}(.)$ is incremented with each event occurrence.

The Probability Density Function (PDF) of a TPP is defined as

$$\mathrm{P}(t|\mathcal{E}(t_i)) = \lim_{\Delta t \to 0} \frac{\mathrm{P}(\text{event } t_{i+1} \text{ in } [t, t+\Delta t]|\mathcal{E}(t_i))}{\Delta t}., \tag{2.2}$$

The Cumulative Distribution Function (CDF) is defined as,

$$\mathrm{F}(t|\mathcal{E}(t_i)) = \mathrm{P}(\text{event } t_{i+1} \text{ in } [t_{i-1}, t]|\mathcal{E}(t_i))., \tag{2.3}$$

The Survival Function is defined as,

$$\mathrm{S}(t|\mathcal{E}(t_i)) = \mathrm{P}(\text{event } t_{i+1} \text{ in } [t, \infty]|\mathcal{E}(t_i)). \tag{2.4}$$

These distributions define the same TPP, and each one can be identified from the other. A convenient and interpretable approach to parameterize PDF is by defining a conditional intensity function (CIF) or instantaneous stochastic rate of events $\lambda(t)$ as defined below,

$$\lambda(t)\mathrm{d}t = \mathrm{E}[\mathrm{d}\mathbb{N}(t)|\mathcal{E}(t)] = \mathrm{P}(\text{event in}[t, t+\mathrm{d}t]|\mathcal{E}(t))\mathrm{d}t, \tag{2.5}$$

where $\lambda(t)\mathrm{d}t$ is the probability of observing an event in interval $[t, t+\mathrm{d}t]$ by observing history till t. From this representation one can see that combination of two independent TPPs with intensities $\lambda_1(t)$ and $\lambda_2(t)$ has the intensity $\lambda(t) = \lambda_1(t) + \lambda_2(t)$ as shown in Figure 2.2.

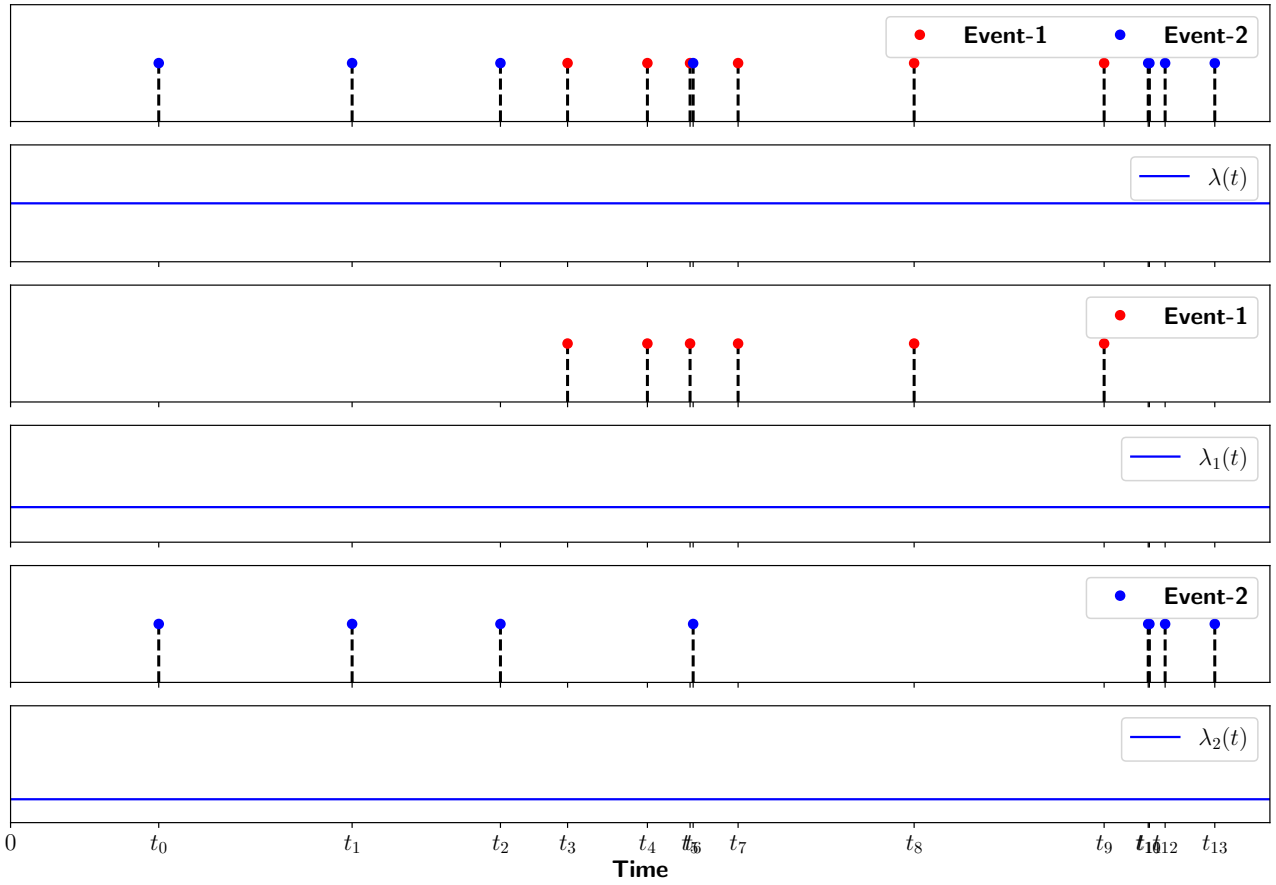


Figure 2.2: Events simulated from two independent Poisson process of intensities $\lambda_1(t)$ and $\lambda_2(t)$ are combined to form a new Poisson process of intensity $\lambda(t) = \lambda_1(t) + \lambda_2(t)$.

One can write $\lambda(t)$ in terms of PDF, $\mathrm{P}(t|\mathcal{E}(t_i))$, and SF, $\mathrm{S}(t|\mathcal{E}(t_i))$ as follows,

$$\begin{aligned}\lambda(t)\mathrm{d}t &= \frac{\mathrm{P}(\text{event in } [t, t+\mathrm{d}t], \mathcal{E}(t))}{\mathrm{P}(\mathcal{E}(t))} = \frac{\mathrm{P}(\text{event } t_{i+1} \text{ in } [t, t+\mathrm{d}t], \mathcal{E}(t_i))}{\mathrm{P}(\mathcal{E}(t_i)S(t|\mathcal{E}(t_i)))}, \\ &= \frac{\mathrm{P}(t|\mathcal{E}(t_i))\mathrm{d}t}{\mathrm{S}(t|\mathcal{E}(t_i))} = \frac{\mathrm{P}(t|\mathcal{E}(t_i))\mathrm{d}t}{1 - \mathrm{F}(t|\mathcal{E}(t_i))}.\end{aligned}$$

The first part of the equation comes from the Bayes rule followed by splitting history $\mathcal{E}(t)$ into intervals $[0, t_i]$ and $[t_i, t]$. Here, probability of events in the interval $[0, t_i]$ is modeled by $\mathrm{P}(\mathcal{E}(t_i))$ and probability that no events observed during the interval $[t_i, t]$ is modeled by a survival function $\mathrm{S}(t|\mathcal{E}(t_i))$. Then one can apply the property $\mathrm{S}(t|\mathcal{E}(t_i)) + \mathrm{F}(t|\mathcal{E}(t_i)) = 1$, which comes from the definition of the survival function and cumulative distribution function. We have the

following,

$$\lambda(t) = \frac{\frac{\mathrm{d}}{\mathrm{d}t}\mathrm{F}(t|\mathcal{E}(t_i))}{1 - F(t|\mathcal{E}(t_i))} = -\frac{\mathrm{d}}{\mathrm{d}t}\log(1 - \mathrm{F}(t|\mathcal{E}(t_i))), \text{and}$$

$$\int_{t_i}^{t} \lambda(t) = -\log(1 - \mathrm{F}(\tau|\mathcal{E}(t_i))).$$

Now one can write, (2.6)

$$\mathrm{F}(t|\mathcal{E}(t_i)) = 1 - \exp\left(-\int_{t_i}^{t} \lambda(t)\mathrm{d}\tau\right),$$

$$\mathrm{S}(t|\mathcal{E}(t_i)) = \exp\left(-\int_{t_i}^{t} \lambda(t)\mathrm{d}\tau\right),$$

$$\text{and } \mathrm{P}(t|\mathcal{E}(t_i)) = \lambda(t)\exp\left(-\int_{t_i}^{t} \lambda(t)\mathrm{d}\tau\right) = \lambda(t)\mathrm{S}(t|\mathcal{E}(t_i)). \tag{2.7}$$

For more details, one can refer to (Daley and Vere-Jones, 2003, 2007).

2.1.1 Marked Temporal Point Process

A TPP event can also be associated with a mark, the feature associated with that event. This type of TPP is called marked temporal point process represented as $\mathcal{E}(t_i) = \{(t_1, e_1), \ldots, (t_i, e_i)\}$ (Jacobsen, 2006). Here, e_i can take categorical ($\mathcal{C} = \{c_1, c_2, \ldots, c_{|\mathcal{C}|}\}$) or real-valued features vectors ($\mathbb{R}^d$, where d is the dimension) or both depending upon the application one is trying to model. For example, in a spatiotemporal point process (Moller and Waagepetersen, 2003, Baddeley et al., 2007) the mark is the coordinates of the location where the event has occurred, and in earthquake models (Ogata, 1988, Omi et al., 2014) mark is the intensity of the earthquake. In the sequel, we will use the term TPP for both marked and unmarked TPP. In this case, the conditional intensity function is written as,

$$\lambda_e(t) = \lambda(t)\mathrm{P}(e|\mathcal{E}(t)), \text{and PDF of the TPP is written as,}$$

$$\mathrm{P}(t, e|\mathcal{E}(t_i)) = \lambda(t)\exp\left(-\int_{t_i}^{t} \lambda(\tau)\mathrm{d}\tau\right)\mathrm{P}(e|\mathcal{E}(t)). \tag{2.8}$$

2.1.2 Likelihood

Given that events are observed during the following times $\{t_1, \ldots, t_i, \ldots t_n\}$ in the interval $[0, T]$, the likelihood of the process can be written as follow,

$$\mathcal{L} = \mathrm{P}(\{t_1, \ldots, t_i, \ldots t_n\}) = \mathrm{P}(t_1|\mathcal{E}(t_0))\mathrm{P}(t_2|\mathcal{E}(t_1)))\ldots\mathrm{P}(t_n|\mathcal{E}t_{n-1})\mathrm{S}(T|\mathcal{E}(t_n)). \tag{2.9}$$

Here, the survival function is the probability that no event has happened since the last time t_n to end time T. Then using Equation 2.6 one can show that,

$$\mathcal{L} = \left(\prod_{i=1}^{n} \lambda(t_i)\right) \exp\left(-\int_0^T \lambda(\tau)\mathrm{d}\tau\right). \tag{2.10}$$

Similarly, one can derive the likelihood function for the marked TPP $\mathcal{E}(t_n) = \{(t_i, e_i)\}_{i=1}^n$ in the interval $[0, T]$ as follow,

$$\begin{aligned}
\mathcal{L} =& \mathrm{P}(\{(t_1, e_1), \ldots, (t_i, e_i), \ldots, (t_n, e_n)\}) \\
=& \mathrm{P}(t_1|\mathcal{E}(0))\mathrm{P}(e_1|t_1, \mathcal{E}(0))\mathrm{P}(t_2|\mathcal{E}(t_1))\mathrm{P}(e_2|t_2, \mathcal{E}(t_1)) \ldots, \\
& \qquad \mathrm{P}(t_n|\mathcal{E}(t_{n-1}))\mathrm{P}(e_n|t_n, \mathcal{E}(t_{n-1}))\mathrm{S}(T|\mathcal{E}(t_n)) \\
=& \left(\prod_{i=1}^{n} \mathrm{P}(e_i|\mathcal{E}(t_i))\right) \left(\prod_{i=1}^{n} \lambda(t_i)\right) \exp\left(-\int_0^T \lambda(\tau)\mathrm{d}\tau\right) \\
=& \left(\prod_{i=1}^{n} \lambda_{e_i}(t_i)\right) \exp\left(-\int_0^T \lambda(\tau)\mathrm{d}\tau\right).
\end{aligned} \tag{2.11}$$

2.2 Commonly used TPP Models

In this section, we will describe the popular form of TPP models in their univariate form. Based on their functional form of conditional intensity functions, we have categorized these into homogeneous and non-homogeneous Poisson processes.

2.2.1 Poisson Process

The homogeneous Poisson process or Poisson process is a simple form of TPP with a constant conditional intensity, $\lambda(t) = \lambda$, and one can specify this as,

$$\begin{aligned}
\mathrm{P}(t|\mathcal{E}(t_i)) &= \lambda \exp\left(-\lambda(t - t_i)\right), \\
\mathrm{F}(t|\mathcal{E}(t_i)) &= 1 - \exp\left(-\lambda(t - t_i)\right), \\
\text{and } \mathrm{S}(t|\mathcal{E}(t_i)) &= \exp\left(-\lambda(t - t_i)\right).
\end{aligned} \tag{2.12}$$

Here, t_i is the last event time and $t > t_i$. Figure 2.2 shows examples of events generated by Poisson processes with respect to their constant intensity functions. Some of the important properties of the Poisson process are listed below.

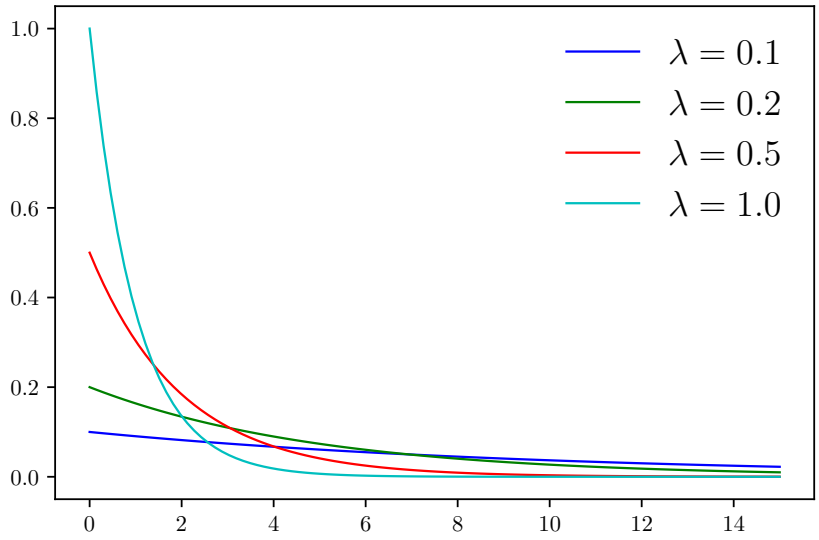


Figure 2.3: Probability density function for Poisson process for different values of λ.

- Inter-event times, $\{(t_1 - t_0), \ldots, (t_n - t_{n-1})\}$, are independent and follow an exponential distribution with rate λ, with expected interevent time, $\mathrm{E}[t_i - t_{i-1}] = \frac{1}{\lambda}$. Figure 2.3 show the PDFs of Poisson processes with different values of λ.
- The number of events in an interval $[t_0, t_n]$ follows a Poisson distribution with rate $\lambda(t_n - t_1)$.
- Given the number of events, n, the arrival times are uniformly distributed in the interval $[t_1, t_n]$.

2.2.2 Non-homogeneous Poisson Process

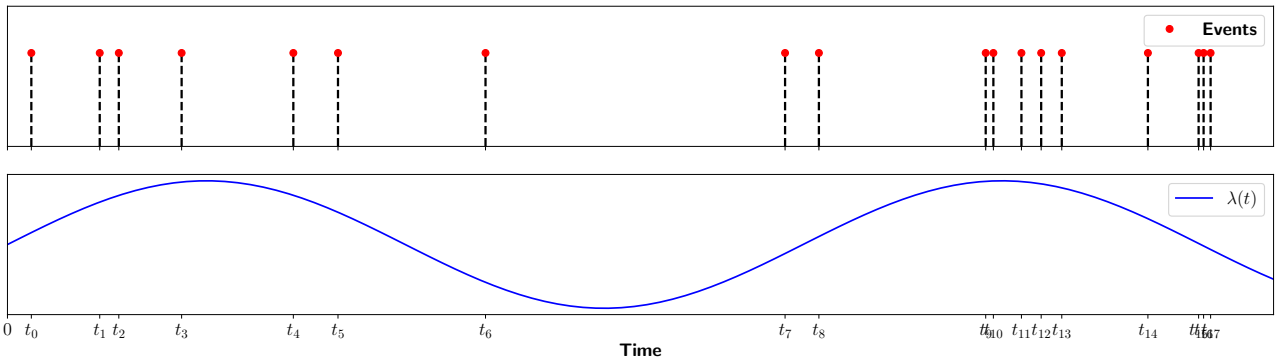


Figure 2.4: Events generated from non-homogeneous Poisson process and its intensity function

Figure 2.4 illustrates an example of events generated from a non-homogeneous Poisson process, where their intensity function changes as a continuous function of time. In the popular versions of non-homogeneous Poisson processes, the change in intensity is modeled as a function of history with discontinuities at the time of occurrence of events.

We will begin with models that have a simple form of conditional intensity functions, which make very restrictive assumptions of the nature of the process to the neural network based methods that are highly flexible. However, this increased flexibility makes it challenging to derive closed analytical forms for survival and moment functions of the distribution. Instead, these functions are estimated using simulation-based approaches, which are explained in Section 2.4.

In addition to the models mentioned above, there are several other variants of Temporal Point Processes (TPPs) that capture specific properties of event generation. Examples include Reinforced Poisson processes (Pemantle, 2007), Reactive point process (Ertekin et al., 2015), and Cox process (Cox, 1955).

2.2.2.1 Rayleigh Process

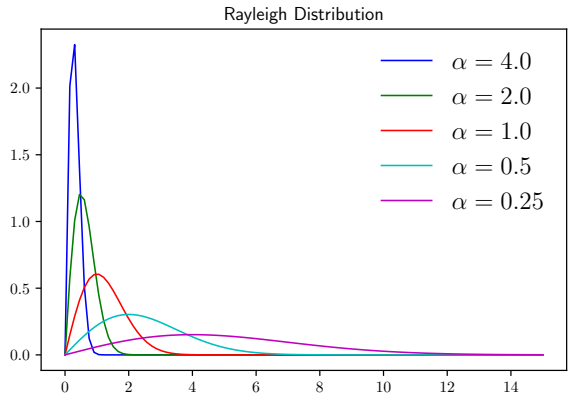


Figure 2.5: Probability density function of Rayleigh distribution for different values of α.

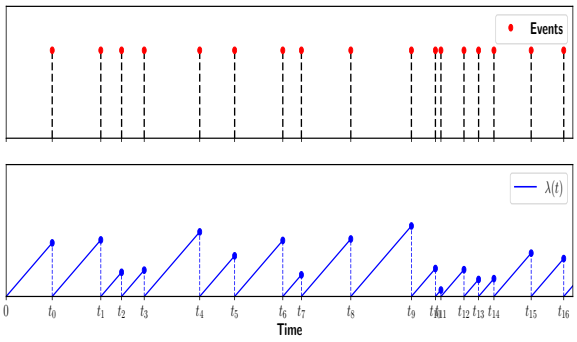


Figure 2.6: Events generated from Rayleigh process and its intensity function.

Rayleigh process has its conditional intensity function as $\lambda(t) = \alpha(t - t_i)$, where the rate of event increasing with time and $\alpha > 0$ is the weight parameter (Ghosh, 2009). Here, t_i is the

last event time. The form probability distribution functions are,

$$\begin{aligned} \mathrm{P}(t|\mathcal{E}(t_i)) &= \alpha(t-t_i)\exp\left(-\frac{\alpha(t-t_i)^2}{2}\right), \\ \mathrm{F}(t|\mathcal{E}(t_i)) &= 1-\exp\left(-\frac{\alpha(t-t_i)^2}{2}\right), \\ \text{and } \mathrm{S}(t|\mathcal{E}(t_i)) &= \exp\left(-\frac{\alpha(t-t_i)^2}{2}\right). \end{aligned} \tag{2.13}$$

Some properties of Rayleigh processes are listed below.

- The inter-event times are independent as the process restarts with each event, as shown in Figure 2.6.
- The PDF drops rapidly after attaining peak as shown in Figure 2.5; hence it is useful for modeling events that are occurring as clusters. Figure 2.6 shows an example of an event sequence simulated using the Rayleigh process and its intensity function. Here, the dashed line indicates the discontinuity at the time of an event.
- Next event duration has the following closed-form,

$$\mathrm{E}[t-t_i] = \sqrt{\frac{\pi}{2\alpha}}. \tag{2.14}$$

2.2.2.2 Self-Correcting Process

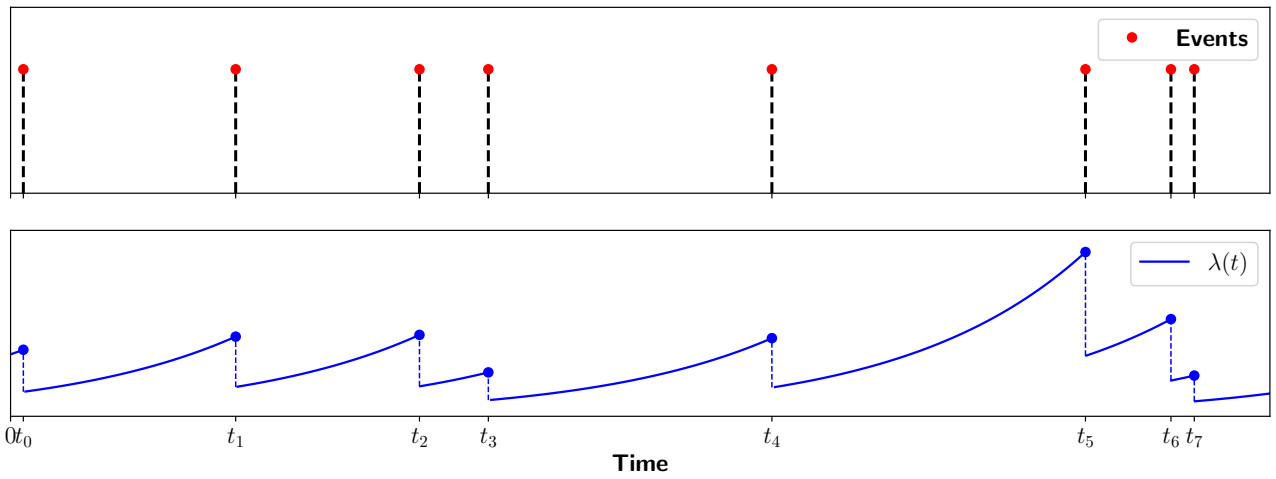


Figure 2.7: Events generated by Self-Correcting process, and the corresponding conditional intensity function. Here, the dashed line indicates the discontinuity at the time of an event.

In the Self-Correcting process (Isham and Westcott, 1979, Xu et al., 2017) the probability of the future event decreases following the occurrence of an event via following intensity function,

$$\lambda(t) = \exp\left(\mu t - \sum_{t_i \in \mathcal{E}(t)} \alpha\right), \tag{2.15}$$

with, $\mu, \alpha \geq 0$. Here, the intensity increases with time but decreases by a factor of $\exp(-\alpha)$ when an event has occurred. Figure 2.7 shows an event sequence generated by a Self-correcting process, and we can observe that the value of the intensity function drop after an event. This is useful for modeling events that are separated in time.

2.2.2.3 Hawkes Process

This is also called the self-exciting process as the probability of future events increases for a short duration with the occurrence of each new event (Hawkes, 1971). It is the most popular form of statistical temporal point process with a wide variety of applications due to its incorporation of history into the conditional intensity function using a kernel function as shown below,

$$\lambda(t) = \mu(t) + \sum_{t_i \in \mathcal{E}(t)} \mathcal{K}(t - t_i). \tag{2.16}$$

Here, $\mu(t) \geq 0$, is the background intensity rate function, kernel function $\mathcal{K}(\cdot) : \mathbb{R}^+ \to \mathbb{R}^+$ quantifies the influence of past events on the current intensity. The kernel function can take different forms like exponential ($\omega \exp(-\omega t)$), Gaussian ($a \exp(-t/\beta)^2$), and power-law ($\frac{a}{(t+c)^{-p}}$) depending upon the nature of the process we are trying to model. Here, ω, a, β, c, p are the parameters of the kernel. One can also improve the representation capacity of the Hawkes process by choosing the background intensity function and kernels as non-parametric functions of history (Lewis and Mohler, 2011, Zhou et al., 2013, Zhang et al., 2020, Zhou et al., 2020).

Hawkes process can also be viewed as a Poisson cluster process (Hawkes and Oakes, 1974) based on superposition principle with each component in the Equation 2.16 as a separate non-homogeneous Poisson process. Here, events are categorized into immigrants if they are generated by the background intensity function $\mu(t)$ and offspring if they are generated by $\mathcal{K}(t - t_i)$. Figure 2.7 illustrates an event sequence and the corresponding intensity function generated by a Hawkes Process. Here, one can see the self-exciting nature of Hawkes, as the conditional intensity function increases instantaneously with each occurrence of the event.

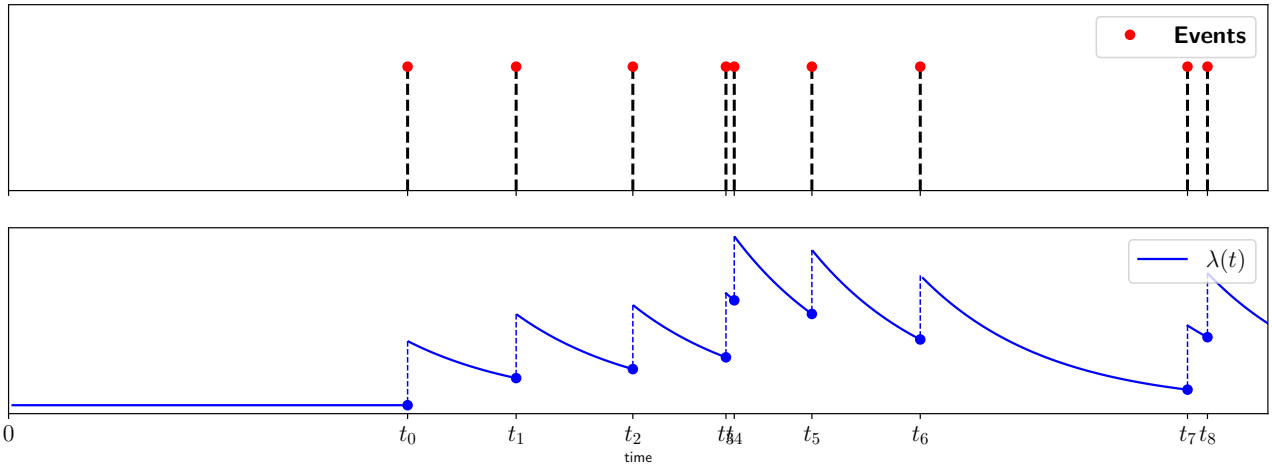


Figure 2.8: Events generated by Hawkes process, and the corresponding conditional intensity function. Here, the dashed line indicates the discontinuity at the time of an event

The PDF of the Hawkes Process can be calculated based on Equation 2.6. However, analytical integration can be challenging, particularly when dealing with complex kernel functions. As a result, there is no closed-form solution available for calculating moments of the distribution, such as the mean duration for the next event. Simulation-based techniques are commonly employed to estimate these quantities and will be explained in Section 2.4. A more detailed analysis of the Hawkes process is provided in the tutorial (Rizoiu et al., 2017).

2.2.2.4 Neural Temporal Point Process

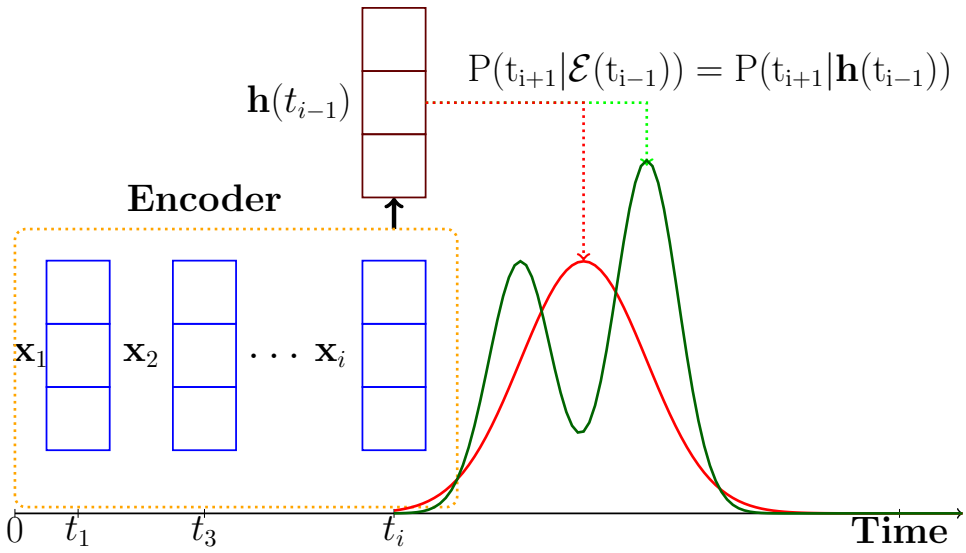


Figure 2.9: Neural TPP block diagram

The earlier mentioned methods use a specific form for conditional intensity function that is crafted based on domain knowledge. However, by exploring deep neural networks as function approximators, one can learn a flexible PDF that is applicable in all domains if sufficient data is provided. In this technique, an auto-regressive neural network takes history as input to get a hidden representation ($\mathbf{h}(t) \in \mathbb{R}^d$) at time t which is then used for parameterizing a TPP. Figure 2.9 shows a Neural TPP block diagram that encodes historical event features $x_i = t_i - t_{i-1}$'s and outputs the PDF for the next event time. These works can be categorized into three types: **(i)** models that parameterize intensity function $\lambda(t)$, **(ii)** models that parameterize compensator function $\Lambda(t) = \int_0^t \lambda(t)\mathrm{d}t$, and **(iii)** models that directly model the PDF $\mathrm{P}(t|\mathcal{E}(t_i))$.

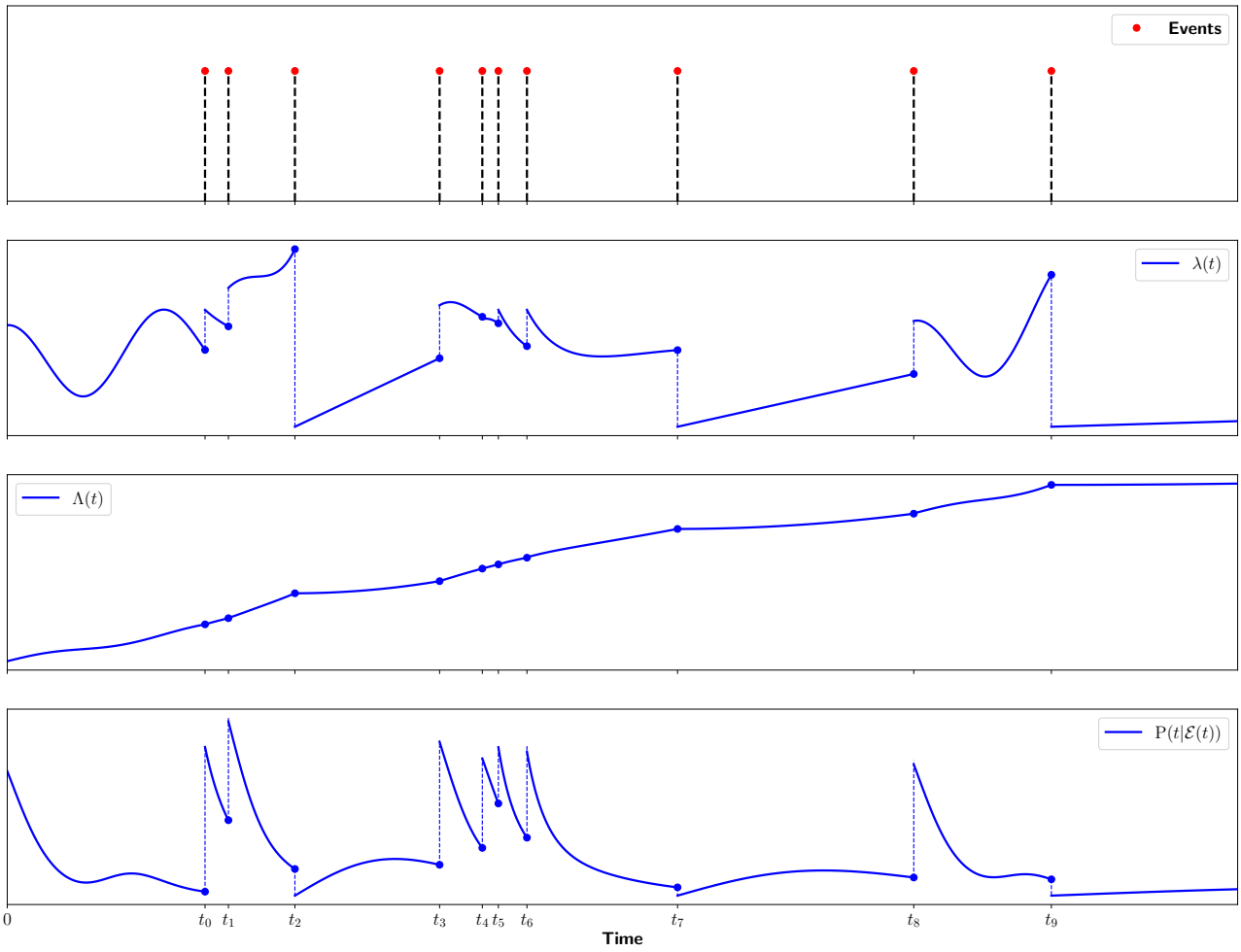


Figure 2.10: Events generated from a Neural TPP and its conditional intensity, compensator, and PDF values. Here, the dashed line indicates the discontinuity at the time of an event

Figure 2.10 shows the event sequence generated using Neural TPP and its conditional intensity, compensator, and PDF values. Here, one can infer that the Neural TPP model is flexible

enough to model complex intensity patterns. Further, we can see the increasing nature of the compensator function with time as it is an integration of the conditional intensity function, which is a positive quantity ($\lambda(t) > 0$). Similarly, we can see the change in PDF after the occurrence of an event as the model adapts to the information from the new event.

Intensity Parameterization. In these types of models, the hidden state is projected into the positive real line followed by a final nonlinear layer $f(.) : \mathbb{R}^d \to [0, \infty)$ that have non-negative range,

$$\lambda(t) = f(\mathbf{W}\mathbf{h}(t)). \tag{2.17}$$

Here, $\mathbf{W} \in \mathbb{R}^{d \times 1}$ is a parameter of the final layer. In Recurrent Marked Temporal Point Processes (Du et al., 2016) uses neural network to encode history $\mathcal{E}(t)$ as follows,

$$\begin{aligned} \lambda(t) &= \text{Softplus}(\mathbf{W}\mathbf{h}(t_i) + \alpha(t - t_i)), \\ \mathbf{h}(t) &= \mathbf{h}(t_i) \text{ Here, } t_i \text{ last event time,} \\ \text{and } \mathbf{h}(t_i) &= \text{RNN}(\mathbf{h}(t_{i-1}), t_i - t_{i-1}). \end{aligned} \tag{2.18}$$

Here, $\alpha \in \mathbb{R}$ is the parameter of the model.

Neural Hawkes Process (Mei and Eisner, 2017) uses a continuous time LSTM model to aggregate history as follows,

$$\begin{aligned} \lambda(t) &= \text{Softplus}(\mathbf{W}\mathbf{h}(t)), \\ \mathbf{h}(t) &= \overline{\mathbf{c}}_i + (\overline{\mathbf{c}}_i - \mathbf{c}_i) \exp\left(-\boldsymbol{\delta}_i(t - t_i)\right), \\ \text{and } \overline{\mathbf{c}}_i, \mathbf{c}_i, \boldsymbol{\delta}_i &= \text{LSTM}(\mathbf{h}(t^i), \overline{\mathbf{c}}_{i-1}, \mathbf{c}_{i-1}). \end{aligned} \tag{2.19}$$

Here, $\overline{\mathbf{c}}_i, \mathbf{c}_i \in \mathbb{R}^d$ are hidden states of continuous LSTM with $\mathbf{c}_i$ as initial state and $\overline{\mathbf{c}}_i$ as the final state. The continuous hidden state $\mathbf{h}(t)$ is an interpolation between the initial state and final state based on the decay function $\exp\left(-\boldsymbol{\delta}_i(t - t_i)\right)$.

Recent works used attention architecture for aggregating history instead of recurrent neural networks. In Transform Hawkes Process (Zuo et al., 2020) use attention architecture with conditional intensity function of Equation 2.18. Similarly, Self Attentive Hawkes Process (Zhang et al., 2020) uses similar attention architecture with conditional intensity function of Equation 2.19. Further, in these models, the integration in the survival function does not have an analytical closed form. Therefore, Monte Carlo integration (Robert and Casella, 2004) is used

as shown below,

$$\int_{t_{i-1}}^{t_i} \lambda(\tau)\mathrm{d}\tau = (t_i - t_{i-1})\frac{1}{L}\sum_{u_i} \lambda(u_i). \tag{2.20}$$

Here, $u_i \sim \text{Uniform}(t_{i-1}, t_i)$ and L is the number of samples used to approximate the integration, and to get an accurate integration higher value of L is needed.

Another line of works model hidden state dynamics based on Neural ODE (Chen et al., 2018, Jia and Benson, 2019, Kidger et al., 2020, Chen et al., 2021). These models employ a full-connected neural network that takes the hidden state at time t as the input and outputs the gradient of that time instant. Integration techniques using ODE solvers such as (Runge, 1895, Kutta, 1901) and (Hairer et al., 1993) are applied to calculate the hidden state at time t as shown below,

$$\begin{aligned} \frac{\mathrm{d}\mathbf{h}(t)}{\mathrm{d}t} &= f_h(\mathbf{h}(t)), \\ \text{and } \mathbf{h}(t) &= \mathbf{h}(t_{i-1}^+) + \int_{t_{i-1}}^{t_i} \frac{\mathrm{d}\mathbf{h}(t)}{\mathrm{d}t}, \end{aligned} \tag{2.21}$$

Here, f_h is the Multi-Layer perceptron function that calculates the gradient of the hidden state, and $\mathbf{h}(t_{i-1}^+)$ is the hidden state after the previous event at time t_{i-1}. Further, when an event occurs, the hidden state is updated using a recurrent neural (Updater) as shown below,

$$\mathbf{h}(t_{i-1}^+) = \text{Updater}(\mathbf{h}(t_{i-1}), x_{i-1}). \tag{2.22}$$

Here, x_{i-1} is the mark associated with the event. To calculate gradients for the model's parameters, integration in the reverse direction is required. This is accomplished using the adjoint sensitivity method (Pontryagin, 1987), which efficiently handles the reverse integration process while minimizing memory usage.

Compensator Function Parameterization. The main problem with neural network based methods is that calculating log-likelihood involves evaluation of,

$$\Lambda(t) = \int_0^t \lambda(\tau)\mathrm{d}\tau. \tag{2.23}$$

Here, $\Lambda(t)$ is the compensator function, and it has to be approximated using Monte-Carlo methods or Neural ODE solvers. Both of these are prone to errors and computationally expensive. Alternately, we can directly parameterize compensatory function based on neural network and

use it for calculating probability based on the following property,

$$\begin{aligned} \lambda(t) &= \frac{\mathrm{d}}{\mathrm{d}t}\big(\Lambda(t)\big) \\ \mathrm{P}(t|\mathcal{E}(t_i)) &= \frac{\mathrm{d}}{\mathrm{d}t}\big(\Lambda(t)\big) \exp\Big(-(\Lambda(t) - \Lambda(t_i))\Big). \end{aligned} \tag{2.24}$$

Recently, (Omi et al., 2019) modeled the compensator using recurrent architecture by creating a function that is monotonously increasing over time t ($\lambda(t) \geq 0$) as shown below,

$$\Lambda(t) - \Lambda(t_i) = \text{Softplus}(\text{MLP}([\mathbf{h}(t_i), t - t_i])). \tag{2.25}$$

However, this model is flawed as it does not define a valid probability distribution that integrates into one. Further, similar to earlier models, simulating future samples is computationally expensive. These have been addressed in the recent work (Shchur et al., 2020) using Normalizing flows (Durkan et al., 2019) based on an invertible function transformations as below,

$$\Lambda(t) = f_1 \circ \ldots f_l \ldots \circ f_L(\mathcal{E}(t)). \tag{2.26}$$

Here, $f_1 \circ \ldots f_l \ldots \circ f_L$ are invertible, (f_l^{-1} exists) spline functions based on Triangular Maps (Jaini et al., 2019). This is defined based on the random time change principle that any TPP can be transformed into Poisson process (Brown et al., 2002). The sampling is done by simulating a Poisson process of unit rate $\mathcal{Z} = \{z_1, \ldots, z_m\}$ and applying inverse transform $\mathcal{E} = f_L^{-1} \circ \ldots \circ f_1^{-1}(\mathcal{Z})$. However, the representation capabilities of these models are limited as they are based on the parametric splines, and also, these models are not defined to incorporate marks.

Probability Density Parameterization. Earlier mentioned models can be used to learn a flexible probability distribution; however, it is difficult to simulate new samples using those distributions. Alternatively, one can parameterize the probability density function in such a way that one can generate samples easily. In (Xiao et al., 2017), the time is modeled using Gaussian distribution with mean as the output of the final layer as shown below,

$$\mathrm{P}(t|\mathcal{E}(t_i)) = \mathrm{N}(t; f(\mathbf{W}\mathbf{h}(t_i)), s^2). \tag{2.27}$$

Here, s^2 is the hyper-parameter of the model. However, this formulation assumes inter-event time follows a Gaussian distribution. In work (Shchur et al., 2020), the inter-event time is

modeled using a more flexible mixture of log-normal distribution as below,

$$\mathrm{P}(t|\mathcal{E}(t_i)) = \sum_{k}^{K} w_k \mathrm{N}(\log(t); \mu_k, s_k^2) \tag{2.28}$$

Here, parameters of distribution, $\{\mu_k, s_k\}_{k=1}^{K}$ are the generated based on the hidden state $\mathbf{h}(t_i)$. The advantage of this formulation is that moments of this distribution can be calculated based on the individual components (Frühwirth-Schnatter, 2006).

A recent trend is to use normalizing flow-based methods as they have been used successfully in the state-of-the-art generative models (Kobyzev et al., 2020). These methods work on the principle that a flexible distribution can be made by transforming a simple distribution. They use the change of variable property of continuous random variable: if $z = g^{-1}(x)$ and $z \sim \mathrm{Q}$, then $\mathrm{P}(x) = |\frac{\mathrm{d}}{\mathrm{d}x} g^{-1}(x)| Q(g^{-1}(x))$. For getting flexible distribution, one can stack with multiple layers $g^{-1} = (g_1^{-1} \circ \ldots \circ g_L^{-1})$ and estimate the PDF as,

$$\mathrm{P}((t|\mathcal{E}(t_i)) = \prod_{l=1}^{L} |\frac{\mathrm{d}}{\mathrm{d}t} g_l^{-1}(t - t_l)| \mathrm{Q}(g^{-1}(t - t_l)). \tag{2.29}$$

Here, Q is the base distribution that is easier to evaluate and sample. Recent work Shchur et al. (2020) proposed two different forms of flow model based on the form of g_l^{-1}, the deep sigmoidal flow ($g_l^{-1} = f_l^{DSF}(x)$) (Huang et al., 2018) and sum-of-squares ($g_l^{-1} = f_l^{SOS}$) (Jaini et al., 2019) as shown below,

$$\begin{aligned} f_l^{DSF}(x) &= \sigma^{-1}\left(\sum_{k=1}^{K} w_k \sigma(\frac{x - \mu_k}{s_k})\right) \\ f_l^{SOS}(x) &= a_0 + \sum_{k=1}^{K}\sum_{p=0}^{R}\sum_{q=0}^{R} \frac{a_{p,k} a_{q,k}}{p + q + 1} x^{p+q+1}. \end{aligned} \tag{2.30}$$

Here, $\{\mu_k, s_k, w_k\}_{k=1}^{K}$, $\{\{\{a_{p,k}a_{q,k}\}_{k=0}^{K}\}_{p=0}^{R}\}_{q=0}^{R}$, and a_0 are the parameters of the function generated from hidden state, $\mathbf{h}(t_i)$ and $\sigma(x) = \frac{1}{1+\exp -x}$. Although calculating likelihood can be done in closed form using the above equation, sampling from it required iterative rooting finding as transformation function g_l^{-1} is not invertible. This is addressed in the recent works Lin et al. (2022), which propose a diffusion flow (Ho et al., 2020) based model that uses variational inference based sampling and likelihood evaluation. They also propose a Continuous Normalizing Flow (Grathwohl et al., 2019) based model that uses Neural-ODE to define a flexible and invertible transform of a random variable.

2.3 Inference

Having discussed the various representative forms of temporal point processes (TPPs), we now delve into the inference of parameters for a general TPP model, including the Neural TPP. At the same time, specialized techniques exist for specific TPP forms like the Hawkes process, such as Expectation Maximization based algorithms, which have been used for parameter estimation in both parametric and non-parametric Hawkes process (Lewis and Mohler, 2011). The work by Rasmussen (2013) used MCMC based techniques to do Bayesian inference of parameters of Hawkes process by using its branching structure.

Maximum Likelihood Estimation (MLE). We use two forms of losses for MLE based parameter estimation depending on the TPP parameterization followed by the Neural TPP model explained in Section 2.2.2.4. For the models that use conditional intensity and compensator parameterization, we use the following negative log-likelihood in Equation 2.10,

$$\arg\max_{\Theta} \mathcal{NLL} = -\sum_{i=1}^{n} \log \lambda(t_i) + \left(\int_0^T \lambda(\tau) \mathrm{d}\tau \right). \tag{2.31}$$

Here, Θ contains the parameters to model the conditional intensity function. In the case of probability density function parameterization, we use its negative log-likelihood as below,

$$\arg\max_{\Theta} \mathcal{NLL} = -\sum_{i=1}^{n} \mathrm{P}(t_i | \mathcal{E}(t_{i-1})). \tag{2.32}$$

Adversarial Training. However, calculating MLE is not always tractable, and it is prone to mode dropping as it is equivalent to minimizing KL divergence between data and model distribution. GAN's (Goodfellow et al., 2014) provide an alternative to likelihood based training by using a discriminator, $\mathrm{Disc}(\cdot)$, to provide supervision to a generator model, $\mathrm{Gen}(\cdot)$. Wasserstein loss (Arjovsky et al., 2017) for temporal point processes is defined as follows,

$$\max_{\Theta_d} \min_{\Theta_g} \mathrm{E}_{\mathcal{E}(T) \sim \mathrm{P}(.))} \left[\mathrm{Disc}(\mathcal{E}(T)) \right] - \left[\mathrm{E}_{\mathcal{Z}(T) \sim \mathrm{Q}(.))} \mathrm{Disc}(\mathrm{Gen}(\mathcal{Z}(T))) \right]. \tag{2.33}$$

Here, the generator model $\mathrm{Gen}(\cdot)$ is used to convert the samples from the noise distribution $\mathcal{Z}(T) \sim \mathrm{Q}(.)$ into the samples from the observed data distribution $\mathcal{E}(T) \sim \mathrm{P}(.)$. This is achieved by using an adversarial training strategy where the discriminator is trained to classify between samples from data distribution as true and samples from the generator as fake. In contrast, the generator is trained to deceive the discriminator classification loss so that generated samples

are similar to observed data. This framework of training TPP models is used in (Arjovsky et al., 2017, Xiao et al., 2017, Yan et al., 2018).

Contrastive Training. For TPP, the probability density function estimation can be intractable as the survival function calculation involves an integration. Noise contrastive (Gutmann and Hyvärinen, 2010) techniques have been used for training models where normalization in the probability density function is not tractable computationally. This is an alternative to GAN based techniques which are known for their difficulty in training. Here, training is done by defining a noise distribution to generate fake samples and then using binary classification based loss to discriminate sames from the true distribution (P) from the noise distribution (Q).

In the case of TPP, the noise distribution is another TPP with intensity function λ^q from which sampling is tractable. This is used to generate M sequence of noise TPP $\{\mathcal{E}^m(T)\}_{m=1}^M$ in the interval $[0, T]$ and combined with true event sequence $\mathcal{E}(T)$ for the new sequence. In this, for a event time t given the history the probability of it is a from true sample is $\frac{\lambda(t_i)}{\lambda(t)+M\lambda^q(t_i)}$ and it is from one of the noise sample is $\frac{\lambda^q(t_i)}{\lambda(t)+M\lambda^q(t_i)}$. Then the noise contrastive loss is as follows,

$$\mathcal{L}_{NCE} = \mathbf{E}_{\mathcal{E}(T)\sim \mathrm{P}(.),\{\mathcal{E}^q(T)\}_{m=1}^M\sim \mathrm{Q}(.)}\Big[\sum_{\mathcal{E}(t,t+\mathrm{d}t)=\{(t_i)\}} \log\frac{\lambda(t_i)}{\lambda(t)+M\lambda^q(t_i)} + \sum_{m=1}^{M}\sum_{\mathcal{E}^m(t,t+\mathrm{d}t)=\{(t_i)\}} \log\frac{\lambda^q(t_i)}{\lambda_h(t_i)+M\lambda_h^q(t_i)}\Big]. \tag{2.34}$$

NCE is first applied for TPP in (Guo et al., 2018); however, it is defined for univariate TPP. In (Mei et al., 2020), the authors extended the use of noise contrastive loss for parameter estimation to multivariate point processes with marks. They have also provided theoretical studies on the optimality, consistency, and effectiveness of the parameters learned by training on NCE loss. In noise contrastive loss, the choice of noise distribution plays a crucial role in determining the convergence speed by avoiding sampling trivial noise samples. To address this, (Mei et al., 2020) proposed using a simpler TPP model that is pre-trained on the data to generate synthetic samples, which are then used as noise samples during the NCE training process.

Reinforcement Learning. This provides an alternative to GAN based models as it requires training two recurrent neural network models, one in the discriminator and the other in the generator. In reinforcement learning based TPP training (Li et al., 2018), each event is considered as an action taken at an asynchronous time by an expert defined by a TPP $\pi_\mathcal{E}$. Then the agent policy π_Θ is learned to match the distribution of the expert based on a reward function

$r(t)$ as shown below,

$$\pi_\Theta = \arg\max_{\pi_\Theta} \mathcal{J}(\pi_\Theta) = \mathbf{E}_{\mathcal{E}(T)\sim\pi_\Theta} \sum_{t_i \in \mathcal{E}(T)} r(t_i). \tag{2.35}$$

Here, the reward function is unknown and is learned based on inverse reinforcement learning (Ng and Russell, 2000, Abbeel and Ng, 2004). The training of this model is similar to GAN model, where the agent is trying to simulate expert policy, and the reward function is used to discriminate samples generated by the expert from the ones generated by the agent.

Similar losses have been applied in stochastic control where TPP based agent has to take action to maximize rewards received asynchronously while interacting with an environment based on another TPP. This has been applied to problems such as fake news mitigation (Frühwirth-Schnatter, 2006), smart broadcasting of advertisements (Karimi et al., 2016, Zarezade et al., 2017), and invasive species management (Gupta et al., 2018) to learn a policy that can achieve favorable outcomes with limited expenditure. Earlier models used the Hawkes process for modeling the agent and environment to find an analytical solution for the optimal policy. However, this limited the expressivity of the policy learned, and recent models overcame this by using Neural TPP based models with reinforcement learning for policy learning. TPPRL (Upadhyay et al., 2018) propose a model free algorithm for learning policy from asynchronous rewards with Neural TPP models to generate actions. However, it requires a higher number of samples to converge due to its model free learning strategy. This is addressed in (Qu et al., 2022) by learning the state dynamics while it interacts with the environment.

2.4 Simulation

This section will focus on simulation algorithms applicable to general non-homogeneous Poisson processes. However, it is worth noting that specific simulation algorithms are designed for particular forms of temporal point processes (TPPs). For instance, the Hawkes process can be simulated as independent non-homogeneous processes by representing its branching structure using separate parametric forms for simulating immigrants and offspring events (Møller and Rasmussen, 2005, 2006). Simulating events independently and then combining them using the superposition principle can generate a single stream of events to represent the desired Hawkes process.

Algorithm 1 describes how to simulate event times t_i from probability density function $(\mathrm{P}(t_i|\mathcal{E}(t_{i-1}))$ using the inverse of the cumulative density function. This is based on the principle that the CDF transforms any continuous random variable $\mathbb{R}$ into Uniform in $[0, 1]$ (Devroye, 1986).

Algorithm 1 Simulation by the inverse transform of CDF

Parameters: Cumulative density function $\mathrm{F}(t|\mathcal{E}(t_i))$, and interval length $[0, T]$.
Set $t = 0$, $i = 0$, $\mathcal{E} = \emptyset$.
while $t < T$ **do**:
    Sample $u \sim \text{Uniform}([0, 1])$.
    Next event time $t = \mathrm{F}^{-1}(u|\mathcal{E}(t_{i-1})) + t_{i-1}$.
    If $t < T$, add t in $\mathcal{E}$ and increase counter $i = i + 1$.
end while
return $\mathcal{E}(T) = \{t_1, t_2, \ldots, t_n\}$.

Algorithm 2 is the equivalent version of Algorithm 1 for TPP that is parameterized on conditional intensity or compensator function. This is based on random time change theorem (Brown et al., 2002) that states any sequence $\mathcal{E}(T) = \{t_1, \ldots, t_n\}$ generated based on non-homogeneous Poisson process with intensity $\lambda(t)$ can be converted into a sequence $\mathcal{Z} = \{z_1, \ldots, z_n\}$ generated by a homogeneous Poisson process of rate one on the interval $[0, \Lambda(T)]$ with $z_i = \Lambda(t_i)$.

Algorithm 2 Simulation by the inverse transform of compensator

Parameters: Intensity function $\lambda(t)$ and interval length $[0, T]$.
$t = 0$, $z = 0$, $\mathcal{E} = \emptyset$
while $t < T$ **do**
    Sample $\Delta z_i \sim \text{Exponential}(1)$
    $z = z + \Delta z_i$
    Next event time $t = \Lambda^{-1}(z)$.
    If $t < T$, add t in $\mathcal{E}$ and increase counter $i = i + 1$.
end while
return $\mathcal{E}(T) = \{t_1, t_2, \ldots, t_n\}$.

Algorithm 3 is used for simulating flow-based models explained in Section 2.2.2.4 that parameterize PDF. In this, Q(.) is the base distribution that is easier to sample from, and $\{g_l^{-1}(x)\}_{l=1}^{L}$ are transformation functions that are parameterized by a neural network based on a history encoder. For the flow-based TPP models described in (Shchur et al., 2020), computing an analytical function for $g(x)$ is impossible, hence root finding methods (Ho et al., 2019) have to be used to find the solution for $g_l^{-1}(z_{l+1}) = z_l$. For the diffusion denoising based model, TCDDM (Lin et al., 2022) uses variation distribution to find the function g, and TCCNF (Mehrasa et al., 2019) uses a CNF based that defines a Neural-ODE based invertible transform for g_l^{-1}.

Algorithm 3 Simulation of Flow-based Models

Parameters: transformation functions $\{g_l^{-1}(x)\}_{i=1}^{L}$, Base distribution Q(.) and interval length $[0, T]$.
$t = 0, i = 0, \mathcal{E} = \emptyset$.
while $t < T$ **do**
    Sample $z_1 \sim$ Q(.).
    $l = 1$.
    while $l < L$ **do**
        Find z_{l+1} that satisfies $g_l^{-1}(z_{l+1}) = z_l$.
        $l = l + 1$.
    end while
    $t = t_{i-1} + \Delta z_L$
    If $t < T$, add t in $\mathcal{E}$ and increase counter $i = i + 1$.
end while
return $\mathcal{E}(T) = \{t_1, t_2, \ldots, t_n\}$.

Earlier mentioned algorithms have constraints on their functional form, like invertibility. However, finding analytical solutions to these is not always applicable to popular TPPs like Hawkes Process or Neural TPP. Alternately, one can use Thinning based algorithms that simulate samples from the Poisson process with intensity higher than that of the TPP of interest and then thin out the samples to match them with TPP of interest (Lewis and Shedler, 1979, Ogata, 1981) as shown in Algorithm 4.

Algorithm 4 Ogata thinning algorithm for simulating non-homogeneous Poisson process

Parameters: Intensity function $\lambda(t)$ and interval length $[0, T]$.
$t = 0$, $\mathcal{E} = \emptyset$
while $t < T$ **do**
    Sample $\Delta t_i \sim$ Exponential($\lambda_{max}(t)$)
    Set $t = t + \Delta t_i$
    Sample $u \sim$ Uniform($[0, 1]$).
    if $u \leq \lambda(t)/\lambda_{max}(t)$ **then**
        Add t to $\mathcal{E}$ and $i = i + 1$.
    end if
end while
return $\mathcal{E}(T) = \{t_1, t_2, \ldots, t_n\}$

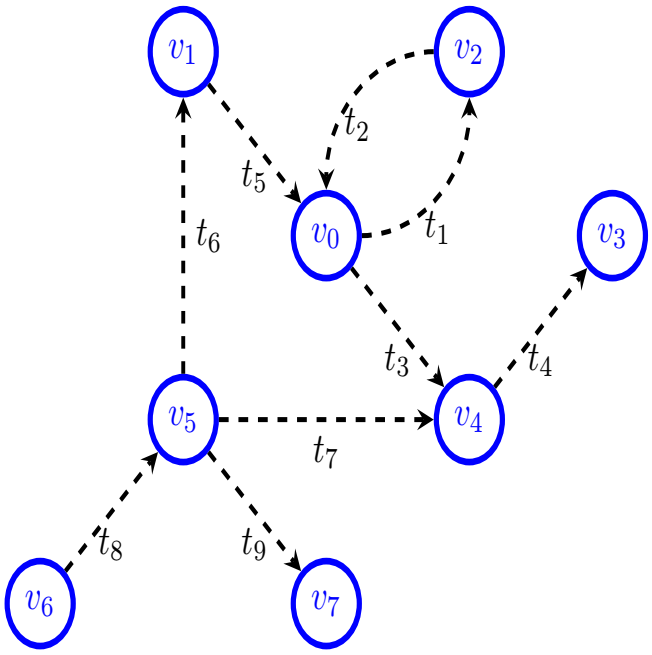


Figure 2.11: Pairwise interaction between nodes in a graph consisting of nodes $\mathcal{V} = \{v_i\}_{i=0}^{7}$. Here, each edge between nodes exists only at the time of interaction.

Here, $\lambda_{max}(t) \geq \max_{t \leq s \leq T} \lambda(s)$, and the thinning is done by accepting samples generated with probability $\lambda(t)/\lambda_{max}(t)$. Even though this algorithm provides a simple alternative for sampling TPP, this way of sampling is computationally expensive if the acceptance probability $\lambda(t)/\lambda_{max}(t)$ is low. In addition to that, finding the upper bound $\lambda_{max}(t)$ is also expensive depending on the form of intensity function.

2.5 Temporal Point Process on Graphs

Here, TPPs are used to model interactions occurring in a graph as events, $\mathcal{G}(t) = (\mathcal{V}, \mathcal{H}, \mathcal{E}(t))$. Here, $\mathcal{H}$ is all the pair-wise nodes in the networks, and $\mathcal{E}(t)$ is the order set of all events that occurred till time t in the network. Figure 2.11 shows a graph of eight nodes and interactions between them as edges along with their time of occurrence. The probability for interaction occurring between nodes v_i and v_j at time t can be modelled as, $\mathrm{P}_{ij}(t|\mathcal{E}(t_n))$. Here, $\mathcal{E}(t_n)$ is historical events till time t_n. The likelihood of the event sequence $\mathcal{E}(T)$ observed during interval $[0, T]$ can be modelled as,

$$\mathcal{L} = \prod_{((v_i,v_j),t_n) \in \mathcal{E}(T)} \mathrm{P}_{ij}(t_n|\mathcal{E}(t_{n-1})) \prod_{(v_i,v_j) \in \mathcal{H}} \mathrm{S}_{ij}(T|\mathcal{E}(t_n)). \tag{2.36}$$

Here, $\mathrm{S}_{ij}(.)$ is the survival function that models the probability of events not occurring between v_i and v_j during the interval $[t_n, T]$. This is similar to a marked TPP with edges as the marks. We can use intensity to parameterize the TPP, and then we can write $\mathrm{P}_{ij}(t_n|\mathcal{E}(t_{n-1})) = \lambda_{ij}(t_n)\exp\left(-\sum_{(v_i,v_j)\in\mathcal{H}}\int_{t_n}^{t}\lambda_{ij}(\tau)\mathrm{d}\tau\right)$ and $\mathrm{S}_{ij}(t|\mathcal{E}(t_{n-1})) = \exp\left(-\int_{t_{n-1}}^{t}\lambda_{ij}(\tau)\mathrm{d}\tau\right)$. Then the likelihood becomes,

$$\mathcal{L} = \prod_{((v_i,v_j),t_n)\in\mathcal{E}(T)} \lambda_{ij}(t_n) \prod_{(v_i,v_j)\in\mathcal{H}} \exp\left(-\sum_{(v_i,v_j)\in\mathcal{H}}\int_0^T \lambda_{ij}(\tau)\mathrm{d}\tau\right). \tag{2.37}$$

The current research in this domain can be categorized into statistical and deep learning based models.

Statistical Models. The works on statistical model define Hawkes processes on a graph where the parameters of the base intensity function and kernels are derived from the node attributes. The general form of this Hawkes process is as shown below,

$$\begin{aligned}\lambda_{ij}(t) = f_\mu(\mathbf{v}_i,\mathbf{v}_j) + \sum_{t_n\in\mathcal{E}_{ij}(t)} \mathcal{K}_0(t-t_n; f_{\mathcal{K}_0}(\mathbf{v}_i,\mathbf{v}_j)) + \sum_{t_n\in\mathcal{E}_{ji}(t)} \mathcal{K}_1(t-t_n; f_{\mathcal{K}_1}(\mathbf{v}_j,\mathbf{v}_i)) \\ + \sum_{t_n\in\mathcal{E}_i(t)} \mathcal{K}_2(t-t_n; f_{\mathcal{K}_2}(\mathbf{v}_i)) + \sum_{t_n\in\mathcal{E}_j(t)} \mathcal{K}_3(t-t_n; f_{\mathcal{K}_3}(\mathbf{v}_j)).\end{aligned} \tag{2.38}$$

Here, $\mathcal{K}_0(\cdot), \mathcal{K}_1(\cdot), \mathcal{K}_2(\cdot), \mathcal{K}_3(\cdot)$ are the kernel functions which can be a standard kernel or a combination of them, and the parameters of these are the functions of node attributes. The node attributes ($\mathbf{v}_i, \mathbf{v}_j \in \mathbb{R}^d$) are learned by minimizing the negative log-likelihood of the events observed. The history sets $\mathcal{E}_{ij}(t)$ and $\mathcal{E}_{ji}(t)$ denote the historical interactions happened between node pairs (v_i, v_j) and (v_j, v_i), respectively. Similarly, history sets $\mathcal{E}_i(t)$ and $\mathcal{E}_j(t)$ are the historical interactions happened on nodes v_i and v_j, respectively. The functions $f_\mu(\cdot)$, $f_{\mathcal{K}_0}(\cdot)$, $f_{\mathcal{K}_1}(\cdot)$, $f_{\mathcal{K}_2}(\cdot)$, and $f_{\mathcal{K}_3}(\cdot)$ output the parameters of the Hawkes Process from the node attributes. Here, $\mathcal{K}_1(\cdot)$ models the influence of reciprocity, and $\mathcal{K}_0(\cdot)$ models the influence of homophily of the interaction between node pair (v_i, v_j). The kernels $\mathcal{K}_2(\cdot)$ and $\mathcal{K}_3(\cdot)$ models the influence of other events occurring on nodes v_i and v_j respectively. The previous works by Yang et al. (2017), Huang et al. (2022), Passino and Heard (2022), parameterize these kernels with latent node attributes to model events. Alternately, works by Blundell et al. (2012), Arastuie et al. (2020), Junuthula et al. (2019), Soliman et al. (2022) uses latent community structure of the nodes as attributes, and $\mathbf{v}_i, \mathbf{v}_j$ are the community identities of the nodes. Here, all the parameters are shared across a community pair; hence the number of parameters needed to be learned reduces considerably. In these, learning is done by an alternate learning strategy of

inferring community structure and event modeling.

Deep Learning based Models. In these models, the continuous embeddings of nodes in the graph are used to parameterize the conditional intensity function as shown below,

$$\lambda_{ij}(t) = f(\mathbf{v}_i(t)\mathbf{v}_j(t)). \tag{2.39}$$

Here, $f(.) : \mathbb{R}^d \times \mathbb{R}^d \leftarrow \mathbb{R}^+$ is an MLP layer that has a positive real line as range, and $\mathbf{v}_i(t), \mathbf{v}_i(t) \in \mathbb{R}^d$ are continuous time nodes embedding for node v_i and v_j as a function of history. The works DeepCoevolve (Du et al., 2016), DyREP (Trivedi et al., 2019), DSPP (Cao et al., 2021), GHNN (Han et al., 2020), and GNPP (Xia et al., 2022) use the above framework to forecast interactions in a network.

2.6 Temporal Point Process and Deep Learning based Covid-19 forecasting for India

This section provides technical details of a deep learning-based tool called CoviHawkes[1] that forecasts India's daily case counts at the national, state, and district levels. The predictions made by our tool may help the policymakers identify vulnerable regions to enact "local" lockdowns proactively. A more detailed version of this section is available in our works (Dukkipati et al., 2021, Adiga et al., 2023).

2.6.1 CoviHawkes Model

In this section, we describe our method for a region of interest $\mathcal{R}$ that can be a district, state, or nation. Let $C(t) \in \mathbb{N}$ denote the number of new Covid-19 cases in the region $\mathcal{R}$ at time t. We use $\mathbf{m}(t) \in \mathbb{R}^{d_m}$ to denote a vector of mobility features for region $\mathcal{R}$ that describes the percentage change in various activities such as "going to work", "staying at home", "grocery shopping", etc, in this region on day t. Additionally, we will use $\mathcal{E}(t)$ to refer to the history of observed data till time t, i.e., $\mathcal{E}(t) = \{(C(s), \mathbf{m}(s))\}_{s \le t}$.

Conditioned on the history $\mathcal{E}(t-1)$, our model assumes that $C(t)$ is a Poisson distributed random variable with mean $\lambda(t)$. Therefore, for all $c = 0, 1, 2, \ldots,$

$$\mathrm{P}\left(C(t) = c \mid \mathcal{E}(t-1)\right) = \frac{\lambda(t)^c \exp(-\lambda(t))}{c\,!}.$$

[1] https://sml.csa.iisc.ac.in/covihawkes

Following the discrete-time Hawkes process literature, we model $\lambda(t)$ as

$$\lambda(t) = \mu + \sum_{i=1}^{L} w(L-i)R(t-i)C(t-i).$$

Here, $\mu \in [0, \infty)$ is the base count rate for the region of interest, L is the length of the time window within which past case counts affect the present case count, $R(t)$ is the *reproduction number* of the virus, and $w(1), \ldots, w(L)$ are weights assigned to the previous L days such that $w(i) \geq 0$ and $\sum_{i=1}^{L} w(i) = 1$.

In other words, infections occur on day t with a base rate of μ. However, a person may also catch the virus from another person infected i days ago ($i \leq L$). The probability of such an infection is proportional to the weight $w(L-i)$ assigned to that day and the reproduction number $R(t-i)$ of the virus i days ago. The parameters μ, $w(1), \ldots, w(L)$, and $R(t)$ are learned directly from the data by maximizing the log-likelihood of the observations. We model $R(t)$ as a function of the past mobility data using an LSTM as described in Section 2.6.1.1.

In the real world, a region has a finite population, and it is rare for people to get infected twice. Moreover, vaccinated people are also less likely to get infected, and hence can be removed from the set of susceptible population members for modeling purposes (Rizoiu et al., 2018). To consider these factors, we use a discounted value of $\lambda(t)$ as shown below:

$$\tilde{\lambda}(t) = \left(1 - \frac{N(t-1) + V(t-1)}{N}\right) \lambda(t).$$

Here, $N(t-1) = \sum_{s=1}^{t-1} C(s)$ is the total number of people that have already been infected before time t, $V(t-1)$ is the number of vaccinated people, and N is the total size of the population in the region. As $N(t-1)$ and $V(t-1)$ increase, the discounted rate $\tilde{\lambda}(t)$ decreases. All parameters, including the parameters of the LSTM that computes $R(t)$, are learned via maximum-likelihood estimation.

2.6.1.1 Estimating the value of $R(t)$

$R(t)$ measures the rate at which the virus reproduces itself. A higher value of $R(t)$ entails a faster spread of the disease and vice versa. Owing to the emergence of variants of the virus, the value of $R(t)$ changes over time and must be estimated periodically using recent trends in the data. To do so, we use an LSTM (Hochreiter and Schmidhuber, 1997). Let $\mathbf{x}(t) \in \mathbb{R}^{(L+1)d_m}$ be a vector obtained by concatenating the mobility vectors $\mathbf{m}(t-i-\Delta)$ for $i = 1, \ldots, L$, and the vector of case counts in the previous L days $[C(t-L), \ldots, C(t-1)]$. The mobility features used at time t at delayed by an additional gap $\Delta > 0$ to take the incubation period of the disease

into account. We use $d_m = 6$, $L = 28$, and $\Delta = 14$ in our experiments. $R(t)$ is then computed as:

$$\mathbf{h}(t) = \text{LSTM}(\mathbf{h}(t-1), \mathbf{x}(t))$$
$$R(t) = \ln(1 + \exp(\mathbf{w}^\intercal \mathbf{h}(t) + b)).$$

Here, $\mathbf{h}(t) \in \mathbb{R}^d$ is the hidden state of the LSTM at time t, and $\mathbf{w} \in \mathbb{R}^d$ and $b \in \mathbb{R}$ are learnable parameters.

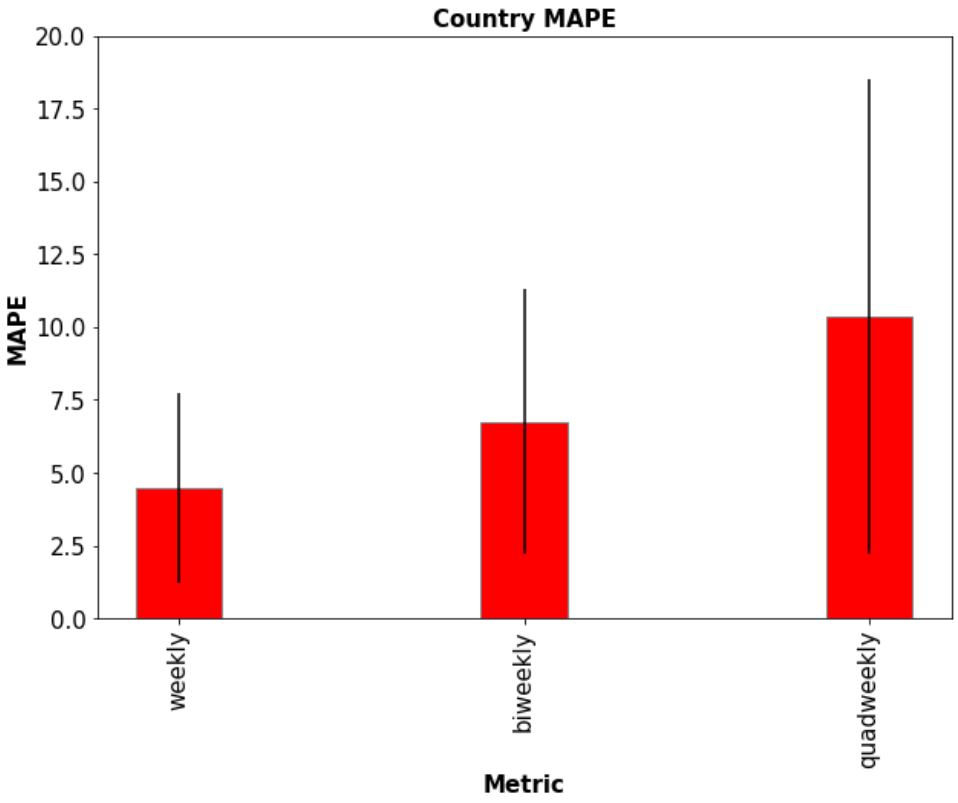
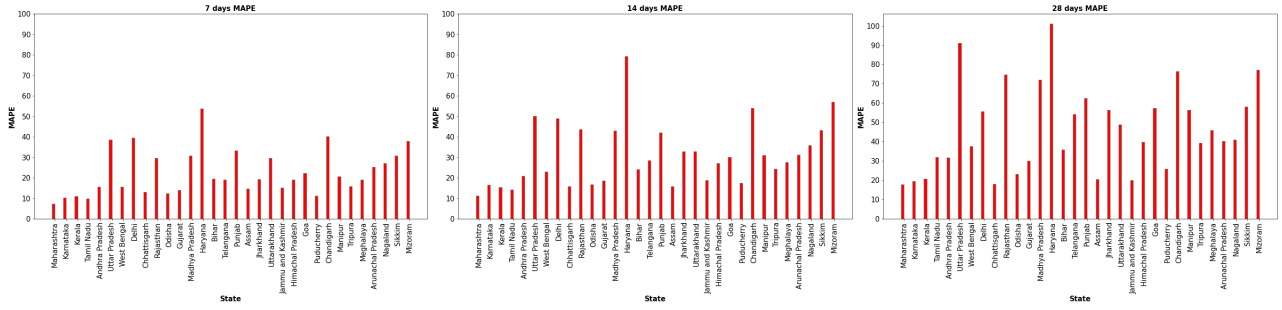


Figure 2.12: Country-Level MAPE scores for different sizes of forecasting window.

2.6.2 Short-term predictions

For validating our short-term predictions, we use case counts $C(t)$ and mobility features $\mathbf{m}(t)$ collected between March 2, 2020, and July 20, 2021. We divide this data into two subsets: training data (March 2, 2020 to April 27, 2021) and validation data (April 27-July 20, 2021), and validate our model for forecasting windows of three sizes: 7, 14, and 28 days.

The validation period has 84 days. To validate the model for a forecasting window of size w, we divide the validation period into n possibly overlapping intervals of size w each. Let



(a) $w = 7$ (b) $w = 14$ (c) $w = 28$

Figure 2.13: MAPE for models trained at the state level for different states and values of forecast window size w.

$I_1^w, I_2^w, \ldots, I_n^w$ be these intervals, then:

$$I_i^w = \{t_s + 7(i-1) + k\}_{k=0}^{w-1},$$

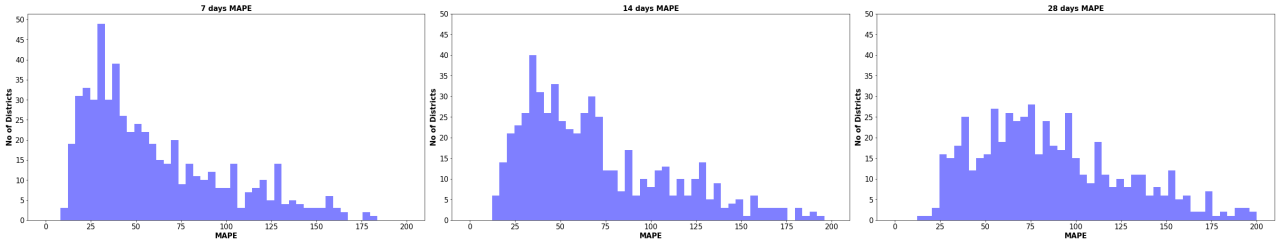
where t_s marks the start of the validation period. To make predictions for interval I_i^w, we independently train a model using all available data till the start of I_i^w and compute our predictions for this interval. Let $\hat{C}_i^w(t)$ be the prediction made by such a model at time t. The error for interval I_i^w is calculated as:

$$\psi(I_i^w) = \frac{|\sum_{t \in I_i^w} C(t) - \sum_{t \in I_i^w} \hat{C}_i^w(t)|}{\sum_{t \in I_i^w} C(t)} \times 100.$$

The error metric above is known as the Mean Absolute Percentage Error (MAPE). The process above is repeated for all $i = 1, \ldots, n$, and the error of the model on a window of size w is given by:

$$E(w) = \frac{1}{n} \sum_{i=1}^{n} \psi(I_i^w).$$

Figure 2.12 shows $E(w)$ for $w = 7$, 14, and 28 for the nation-level model that forecasts case counts for the country as a whole. Figure 2.13 shows the same data for state-level models. As the number of districts is very large (> 500), we use histograms in Figure 2.14 to show the number of districts on the y-axis that have the corresponding MAPE scores on the x-axis. As before, Figure 2.14 has histograms for different values of w. The previous forecasts based on this model are available on our website https://sml.csa.iisc.ac.in/covihawkes.



(a) $w = 7$ (b) $w = 14$ (c) $w = 28$

Figure 2.14: Histogram of MAPE scores for models trained at district level for different values of forecast window size w.

Interval	Allowed	Not Allowed
I_s	Essential government services like defence, police, and power generation. Healthcare systems, related industries, and emergency services. Banks. Shops selling ration, meat, dairy, and animal fodder. Internet services. E-commerce. Media.	All non-essential offices, shops, and industries. All transport (except for essential goods). All educational institutes, places of worship, and gatherings of any kind. Hospitality services.
I_m	All businesses, some at limited capacity. Graded reopening of schools. Cinema/Theaters at 50% capacity. Gatherings of up to 100 people in closed spaces with maximum occupancy of 50%. Transportation.	Large gatherings in indoor spaces. Different states have different restrictions.
I_c	Same as I_m, but with higher mobility values (see Figure 2.15)	
I_n	Everything open, business as usual	

Table 2.1: Lockdown conditions. I_s: Strict lockdown, I_m: Unlock Phase 7, I_c: Current conditions, I_n: No lockdown. See Section 2.6.3 for details about these conditions.

2.6.3 Long-term predictions

Recall that our model uses mobility features $\{\mathbf{m}(t-\Delta-i)\}_{i=1}^{L}$ and case counts $\{C(t-i)\}_{i=1}^{L}$ from the last few days to forecast the future value at time t. Thus, we can only generate a forecast at time t if these feature values are available. To make predictions over a longer time horizon, we need appropriate proxies for the values of these features. The case counts can be boot-strapped, i.e., the model can treat its own past predictions as observed ground truth values. However, simulating the mobility values is much harder. Instead, we take a simplified approach, as explained below.

We first identify four time intervals in the past that correspond to different lockdown conditions. The first interval, I_s = [March 25-April 14, 2020], covers the time when there was a strict nationwide lockdown. The second interval, I_m = [December 13-19, 2020], falls in the seventh *unlock* phase in India where most, though not all, of the restrictions, were lifted. The third interval, I_n = [February 15-March 3, 2020] corresponds to the time before the pandemic when there was no lockdown. Finally, I_c = [August 13-19, 2021], corresponds to the current mobility conditions. Table 2.1 summarizes the conditions under these intervals.

We generate four long-term forecasts, one each for the mobility conditions from the intervals I_s, I_m, I_n, and I_c. To generate forecasts corresponding to I_{x} ($\mathrm{x} \in \{s, m, n, c\}$), we compute the average value of mobility features for each weekday $i = 1, \ldots, 7$ in I_{x}. Let $D(i)$ be the set of all i^{th} weekdays between February 14, 2020 and August 20, 2021. For example, $D(1)$ will be the set of all Sundays in this interval. Then, define $\mathbf{m}_{\mathrm{x}}(i)$ as,

$$\mathbf{m}_{\mathrm{x}}(i) = \frac{1}{|I_{\mathrm{x}} \cap D(i)|} \sum_{t \in I_{\mathrm{x}}} \mathbf{1}\{t \in D(i)\}\mathbf{m}(t).$$

Whenever we require the mobility value for a day t in the future that has not been observed, we use $\mathbf{m}_{\mathrm{x}}(i)$ for an appropriate i chosen based on the day of the week at time t. This, together with bootstrapped count values, enables us to use our model to generate long-term forecasts.

Figure 2.15 shows the average mobility values $\frac{1}{7}\sum_{i=1}^{7}\mathbf{m}_{\mathrm{x}}(i)$ for various lockdown conditions and Figure 2.16 shows the long-term forecast at the nation-level under these conditions. One can see from Figure 2.15 that the current conditions have a higher mobility value as compared to the seventh unlock phase, despite the restrictions being similar in these two phases (see Table 2.1). Consequently, the model predicts a sharper rise in cases under I_c as compared to I_m. One can also see that the model predicts a significant third wave if the mobility returns to the pre-pandemic conditions (I_n). Interestingly, the model also indicates that a mild lockdown (as in I_m) would be almost as good as a very strict lockdown (as in I_s).

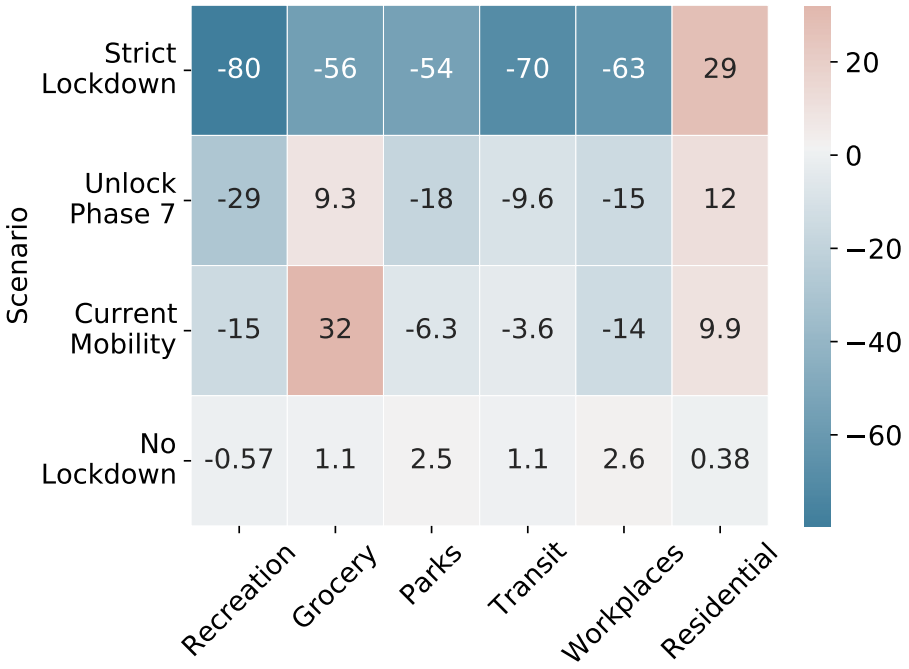


Figure 2.15: Heatmap showing average mobility values under various lockdown conditions. The columns correspond to the six mobility features that we use. Negative values indicate less mobility (the values represent percentage change as compared to a pre-pandemic baseline).

As is the case with every statistical model, CoviHawkes also makes several assumptions. For example, we train models independently for states and districts and do not consider the movement of people across these regions due to the lack of this data. Similarly, the model does not consider breakthrough infections, reinfections, or emergence of new variants of the virus in the long term. While our short-term forecasts are rigorously validated, validating long-term forecasts can be challenging. In particular, long-term forecast uses proxy values for the features as described above, and hence the errors compound over time. Moreover, usage of average mobility features ignores events like festivals and/or other unforeseen mass gatherings that are likely to trigger an uptick in the infection rate. As such, one must keep these caveats in mind while using these long-term predictions.

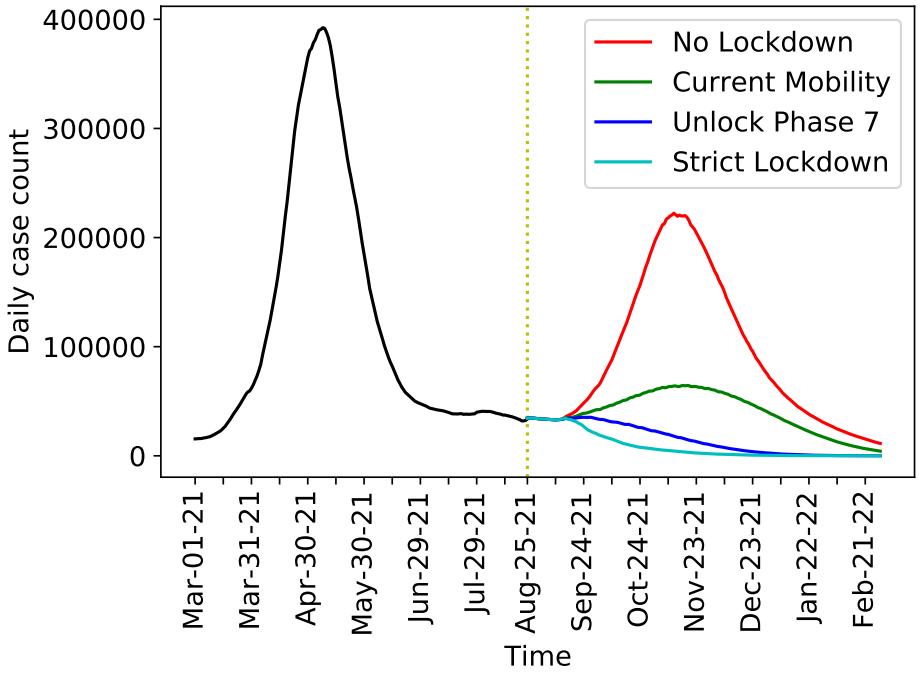


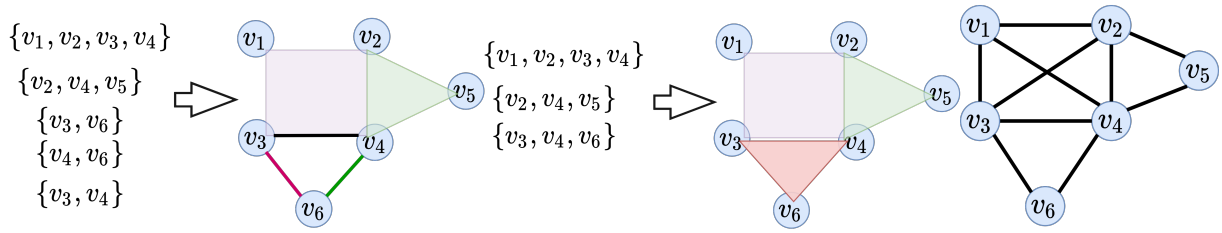
Figure 2.16: Nation-level long-term forecast.

Chapter 3

HyperTPP for Events in Hypergraphs

Most real-world interactions are more complex than just pairwise interactions. For example, a person can have multiple items in a single shopping order, a group of people can co-author an article, mutual funds have stocks of various companies, and so on. These types of problems have been studied using hypergraphs, and there have been many theoretical advances have taken place (Ghoshdastidar and Dukkipati, 2017a,b). A common technique employed to deal with this problem is to approximate these multiway interactions with pairwise interactions, which amounts to approximating hypergraphs with graphs. This leads to enormous information loss, as demonstrated in Figure 3.1. In this example, two different kinds of interaction between nodes $\{v_1, v_2, v_3, v_4, v_5, v_5\}$ have the same pairwise interaction graph. Further, it is impossible to infer the original interactions once they are projected into a pairwise graph.

This chapter addresses the problem of forecasting interactions as hyperedge events using TPP. We define a conditional intensity function on each hyperedge that takes node representations as inputs. Since each hyperedge can have a variable number of nodes, we use a self-attention based link prediction architecture to parameterize the intensity function. Also, nodes evolve as they interact, so it is essential to employ dynamic node representations. In earlier works on dynamic networks (Trivedi et al., 2019, Dai et al., 2016, Cao et al., 2021), node embeddings are updated based on the node embedding of the other node in the interaction. For example, consider edge event (v_a, v_b) occurring at time t. To update node embeddings of v_a, we use the node embeddings of v_b and vice versa. However, hyperedge events have a variable number of nodes (Figure 3.1), so the techniques developed for pairwise edges are not directly applicable here. Hence, we use a self-attention-based encoding for node updates with parameters shared with the hyperlink prediction model to address this. The results presented in this chapter are available in another form in our published work Gracious and Dukkipati (2023).



(a) Hyperedges at time t

(b) Hyperedges at time t

(c) Approximate pairwise graph for Hypergraphs in 3.1a and 3.1b

Figure 3.1: Interactions at time t are shown as hyperedges in Figures 3.1a and 3.1b. Here, hyperedges are represented by geometric shapes with their ends/corners showing the nodes and color showing their identity. One can see two different hypergraphs having the same projected graph in Figure 3.1c. This demonstrate a need for a technique that predicts hyperedges without approximating the hypergraph with a pairwise graph.

Contributions. We propose a model called *Hyperedge Temporal Point Process* (HyperTPP) to model interactions as hyperedge events in a dynamic Hypergraph. Further, one can notice that these interactions will not always be between a homogeneous set of nodes. Hence, we also developed a bipartite hyperedge variant of our model called *Bipartite Hyperedge Temporal Point Process* (BHyperTPP). This will help in modeling interactions between two different types of nodes. Our contributions are as follows.

- A temporal point process framework for hyperedge modeling that can forecast the type and time of interaction.
- A model for representation learning for interaction data.
- Extensive experiments on real-world datasets on both homogeneous and bipartite hyperedges.
- Empirical results on performance gain obtained when we use hyperedges instead of pairwise modeling.
- Empirical results on performance gain are obtained when we use dynamic models instead of static models.

3.1 Dynamic Hyperedge Forecasting

Given a set of nodes $\mathcal{V} = \{v_1, v_2, \ldots, v_{|\mathcal{V}|}\}$, an interaction event between a subset of these nodes is modeled as a hyperedge (h) with the time of interaction as its edge attribute. Here, $h \in \mathcal{H}$ is a subset of nodes in $\mathcal{V}$, and $\mathcal{H}$ is the set of all valid combinations of nodes. Given the historical events $\mathcal{E}(t_n) = \{(e_1, t_1), , \ldots, (e_n, t_n)\}$ till time t_n, here $e_i \in \mathcal{H}$, aim is to forecast future hyperedge e_{n+1} at time $t_{n+1} > t_n$. Table 3.1 summarizes the notations and their definition used in this chapter.

3.1.1 Hyperedge Event Modeling

Given a hyperedge $h = \{v_1, v_2, \ldots, v_k\}$, the probability of h occurring at time t can be modeled using TPP with conditional intensity $\lambda_h(t)$ as

$$\mathrm{P}_h(t) = \lambda_h(t)\mathrm{S}_h(t_h^p, t), \tag{3.1}$$

where $\mathrm{S}_h(t)$ is the survival function that denotes the probability that no event happened during the interval $[t_h^p, t)$ for hyperedge h. This is defined as

$$\mathrm{S}_h(t_h^p, t) = \exp\left(\int_{t_h^p}^{t} -\lambda_h(\tau)\mathrm{d}\tau\right), \tag{3.2}$$

where $t_h^p = \max_{v \in h} t_v^p$ and t_v^p denotes the most recent interaction time of node v in h. The conditional intensity function $\lambda_h(t)$ is parameterized by defining a positive function over the embeddings of nodes in h as,

$$\lambda_h(t) = f(\mathbf{v}_1(t), \mathbf{v}_2(t), \ldots, \mathbf{v}_k(t)). \tag{3.3}$$

Here $f(.) \geq 0$ can be realized by neural network for hyperedge events, and $\mathbf{v}_i(t) \in \mathbb{R}^d$ is the node embeddings at time t for node v_i. We follow the same architecture of the hyperedge modeling technique as Hyper-SAGNN (Zhang et al., 2019) with a final softplus layer as explained in Section 3.1.2. In this paper, we have also developed baseline models using piece-wise constant node embeddings with intensity defined by Rayleigh Process to create an equivalent model for DeepCoevolve (Dai et al., 2016) in hyperedge events, as shown below,

$$\lambda_h(t) = f\Big(\mathbf{v}_1(t_h^p), \mathbf{v}_2(t_h^p), \ldots, \mathbf{v}_k(t_h^p)\Big)(t - t_h^p). \tag{3.4}$$

Notation	Definition
t^p	Previous time
t_h^p	Previous time of interaction for hyperedge h
t^s	Monte-carlo t sample
$\mathrm{P}_h(t)$	PDF of the hyperedge event h
$\mathrm{S}_h(t)$	SF of the hyperedge event h
t_v^p	Previous time of interaction for node v
t_h^l	Last time occurrence of h
$f(\ )$	Scoring function for conditional intensity
$\boldsymbol{\theta}$	Learnable parameter
$\mathbf{v}(t)$	Node embedding at time t
$\mathbf{W}_0, \mathbf{W}_1, \mathbf{W}_2, \mathbf{b}_0$	Learnable parameters for dynamic node embeddings
$\mathbf{W}_3, \mathbf{W}_4, \mathbf{W}_5, \mathbf{b}_1$	Learnable parameters for interaction update
$\mathbf{v}^s(t)$	History aggregation embeddings
$\boldsymbol{\psi}(t)$	Time embedding function
$\{\omega_i\}_{i=1}^d$ and $\{\theta_i\}_{i=1}^d$	Learnable parameters of $\Phi(t)$
$\mathbf{W}_Q, \mathbf{W}_K, \mathbf{W}_V$	Self-Attention learnable weights
γ_{ij}	Attention inner-product score
α_{ij}	Attention weights
$\mathbf{W}_s$	Static embeddings linear layer
$\mathbf{d}_{v_i}^h$	Dynamic embeddings for node v_i in hyperedge h
$\mathbf{s}_{v_i}^h$	Static embeddings for node v_i in hyperedge h
o_i	Output Hadamard layer
$\mathbf{W}_o, b_o$	Output linear layer
$\mathcal{P}$	Scoring function
$\mathcal{K}()$	Kernel function
μ	Baseline intensity
α	Weight parameter
N^q	Negative sample size
N^t	Sampling for Monte-Carlo integration
$\mathcal{B}$	Mini batch segment size

Table 3.1: Notations

This formulation will give us a closed-form solution for survival function, thereby for the probability of the event. Otherwise, integration has to be approximated by sampling.

3.1.2 Conditional Intensity Function Architecture

Given the node embeddings of hyperedge $h = \{v_1, v_2, \ldots, v_k\}$, the importance weights for each node are calculated by a Self-Attention Layer (SAT), as given by,

$$\begin{aligned} \gamma_{ij} &= (\mathbf{W}_Q^T \mathbf{v}_i(t))^T \mathbf{W}_K^T \mathbf{v}_j(t), \forall 1 \leq i, j \leq k, i \neq j, \\ \alpha_{ij} &= \frac{\exp(\gamma_{ij})}{\sum_{1 \leq \ell \leq k, i \neq \ell} \exp(\gamma_{i\ell})}. \end{aligned} \tag{3.5}$$

These weights are used to calculate the dynamic embeddings for each node v_i as,

$$\mathbf{d}_{v_i}^h = \tanh\left(\sum_{1 \leq j \leq k, i \neq j} \alpha_{ij} \mathbf{W}_V^T \mathbf{v}_j(t) \right). \tag{3.6}$$

Here, $\mathbf{W}_Q, \mathbf{W}_K, \mathbf{W}_V \in \mathbb{R}^{d \times d}$ are learnable weights. We also create static embeddings $\mathbf{s}_{v_i}^h = \mathbf{W}_s \mathbf{v}_i(t)$, where $\mathbf{W}_s \in \mathbb{R}^{d \times d}$ is a learnable parameter. Then we calculate the Hadamard power of the difference between static and dynamic embedding pairs followed by a linear layer and average pooling to get the final score $\mathcal{P}^h$ as shown below,

$$\mathcal{P}^h = \frac{1}{k} \sum_{i=1}^{k} \mathcal{P}_i^h = \frac{1}{k} \sum_{i=1}^{k} \log(1 + \exp(o_i)), \qquad o_i = \mathbf{W}_o^T (\mathbf{d}_{v_i}^h - \mathbf{s}_{v_i}^h)^2 + b_o. \tag{3.7}$$

Here, $\mathbf{W}_o \in \mathbb{R}^{d \times 1}, b_o \in \mathbb{R}$ are learnable parameters of output layer. In our model the value of $f(\{\mathbf{v}_1(t), \mathbf{v}_2(t), \ldots, \mathbf{v}_k(t)\}) = \mathcal{P}^h$.

3.1.3 Dynamic Node Representation

For each node in the network, we learn a low dimensional embedding $\mathbf{v}(t) \in \mathbb{R}^d$ that changes with time. It is done through three stages, i) Temporal drift, ii) History aggregation, and iii) Interaction update, as

$$\mathbf{v}(t) = \tanh\Big(\underbrace{\mathbf{W}_0 \mathbf{v}(t_v^{p+})}_{\textit{Interaction Update}} + \underbrace{\mathbf{W}_1 \boldsymbol{\psi}(t - t_v^p)}_{\textit{Temporal Drift}} + \underbrace{\mathbf{W}_2 \mathbf{v}^s(t_{i-1})}_{\textit{History Aggregation}} + \mathbf{b}_0 \Big). \tag{3.8}$$

Here, $\mathbf{W}_0, \mathbf{W}_1, \mathbf{W}_2 \in \mathbb{R}^{d\times d}, \mathbf{b}_0 \in \mathbb{R}^d$ are learnable parameters, $\mathbf{v}(t_v^{p+})$ is the node embedding just after previous interaction for node v at time t_v^p and t_{i-1} is the last event time, $t_{i-1} < t$.

Temporal drift. This term models the inter-event evolution of a node with time. For a node v with previous event time t_v^p, the drift in embedding at time t is modelled by $\mathbf{W}_1\boldsymbol{\psi}(t - t_v^p)$, where $\boldsymbol{\psi}(t) \in \mathbb{R}^d$ is Fourier time features defined as $\boldsymbol{\psi}(t) = [\cos(\omega_1 t + \theta_1), \ldots, \cos(\omega_d t + \theta_d)]$ (da Xu et al., 2020, Cao et al., 2021). Here, $\{\omega_i\}_{i=1}^d$, and $\{\theta_i\}_{i=1}^d$ are learnable parameters.

History aggregation. This stage uses hypergraph convolution-based feature aggregation to incorporate the effect of past events. For this, we will construct a hypergraph using the past $\mathcal{B}$ events, $\{(e_{n-\mathcal{B}}, t_{n-\mathcal{B}}), \ldots, (e_{n-1}, t_{n-1})\}$, and uses its incidence matrix $\mathcal{G}(t_{n-1}, t_{n-\mathcal{B}}) \in \mathbb{R}^{|\mathcal{V}|\times\mathcal{B}}$ to apply hypergraph convolution as,

$$[\mathbf{v}_1^s(t_{n-1}), \ldots, \mathbf{v}_{|\mathcal{V}|}^s(t_{n-1})] = \text{HGNN}\left(\mathcal{G}(t_{n-1}, t_{n-\mathcal{B}}), \left[\mathbf{v}_1(t_{v_1}^{p\,+}), \ldots, \mathbf{v}_{|\mathcal{V}|}(t_{v_{|\mathcal{V}|}}^{p\,+})\right]\right).$$

Here, HGNN is a hypergraph graph convolution defined as in Bai et al. (2021).

Interaction update. When a node v is involved in an interaction h, it is influenced by the nodes it interacts within h. For extracting features of interaction, we use the dynamic embedding $\mathbf{d}_v^h$ calculated as a function of embeddings of nodes $h - \{v\}$ at time t. The architecture of calculating this is shared with conditional intensity function as shown in Equation 3.6 in Section 3.1.2. The entire update equation is $\mathbf{v}(t^+) = \tanh(\mathbf{W}_3\mathbf{v}(t_v^{p+}) + \mathbf{W}_4\boldsymbol{\psi}(t - t_v^p) + \mathbf{W}_5\mathbf{d}_v^h + \mathbf{b}_1)$. Here, $\mathbf{W}_3, \mathbf{W}_4, \mathbf{W}_5 \in \mathbb{R}^{d\times d}$ and $\mathbf{b}_1 \in \mathbb{R}^d$ are learnable parameters. If node v is involved in multiple hyperedge events $\{e_i, e_{i+1}, \ldots e_{i+L}\}$, then we will take the mean of dynamic embeddings from all of the hyperedges. Here, L is the number of concurrent hyperedges.

3.2 Learning Procedure

3.2.1 Loss Function

Once intensity parameterization is fixed for temporal point process as in Equation 3.3 or 3.4, the likelihood for hyperedge events $\mathcal{E}(T) = \{(e_1, t_1), \ldots, (e_N, t_N)\}$ occurring in an interval $[0, T]$ can be modeled as, $\mathrm{P}(\mathcal{E}(T)) = \prod_{n=1}^N \mathrm{P}_{e_n}(t_i) \prod_{h\in\mathcal{H}} \mathrm{S}_h(t_h^\ell, T)$. Here, $\mathrm{P}_{e_n}(t_n)$ is the probability of hyperedge event e_n occurring at time t_i as defined in Equation 3.1, $\mathrm{S}_h(t_h^\ell, T)$ is the probability that no event occurred for hyperedge h for the interval $[t_h^\ell, T]$, and t_h^ℓ is the last time occurrence for h ($t_h^\ell = 0$, when no event of h is observed). The loss for learning parameters can be found by taking the negative of log-likelihood as, $\mathcal{NLL} = -\sum_{n=1}^N \log(\lambda_{e_n}(t_n)) + \sum_{h\in\mathcal{H}} \int_0^T \lambda_h(t)\mathrm{d}t$. Here, the first term corresponds to the sum of the negative log intensity of occurred events. The second term corresponds to the sum of the intensities of all events. The following happens when we

minimize the loss, the intensity rate of occurring events increases due to the minimization of the first term, and the intensity rates of events that do not occur decrease due to the minimization of the second term. However, directly implementing this equation is computationally inefficient as $|\mathcal{H}| \leq 2^{|\mathcal{V}|}$ is very large. Further, the integration in the second term does not always have a closed-form expression. In the next section, we will give this model a computationally efficient mini-batch training procedure.

3.2.2 Mini-Batch Loss

We divide the event sequences into independent segments to make backpropagation through time feasible, as done in previous works on pair-wise dynamic networks (Dai et al., 2016). Then the loss for each segment is calculated as follows, for each (e_i, t_i) in the segment $\mathcal{E}_M = \{(e_1, t_1), \ldots, (e_{\mathcal{B}}, t_{\mathcal{B}})\}$, we use Monte-Carlo integration to find the log survival term of $\mathrm{P}_{e_i}(t_i)$. Then the negative log-likelihood for that event is,

$$\begin{aligned} \mathcal{T}^s = \{t_j^s\}_{j=1}^{N^t} &\leftarrow \text{Uniform}(t_{i-1}, t_i, N^t) \\ \mathcal{NLL}_{e_i} &= -\log(\lambda_{e_i}(t_i)) + \sum_{j=2}^{N^t} (t_j^s - t_{j-1}^s)\lambda_{e_i}(t_j^s). \end{aligned} \tag{3.9}$$

Here, $\mathcal{T}^s$ is the set of uniformly sampled time points from the interval $[t_{i-1}, t_i]$. Then to consider the interactions events $h \in \mathcal{H}$ that were not observed during the above period, we sample some negative hyperedges for each interaction event (e_i, t_i) as described below,

1. Choose the size of negative hyperedge k based on a categorical distribution over hyperedge sizes observed in the training data. Here, parameters of the categorical distribution are learned from the training dataset.

2. Sample $\min(\lceil k/2 \rceil, |e_i|)$ nodes from the hyperedge e_i and rest of the nodes from $\mathcal{V} - e_i$. This strategy will avoid trivial negative samples.

Following the above steps, we sample $\mathcal{H}_i^c = \{h_1^c, \ldots, h_{N^q}^c\}$ negative hyperedges, and for each of them, we calculate the negative log-likelihood for events not happening using Monte-Carlo integration. Then Equation 3.9 becomes,

$$\mathcal{NLL}_{e_i} = -\log(\lambda_{e_i}(t_i)) + \sum_{h \in \mathcal{H}_i^c \cup \{e_i\}} \sum_{j=2}^{N^t} (t_j^s - t_{j-1}^s)\lambda_h(t_j^s).$$

The final mini-batch loss is calculated by summing all $\mathcal{NLL}_{e_i}$ for the events (e_i, t_i) in $\mathcal{E}_M$,

$\sum_{i=1}^{\mathcal{B}} \mathcal{NLL}_{e_i}$. Then the gradients are backpropagated for this loss, and the above training procedure is repeated in the next segment.

3.3 Extending to Bipartite Hyperedges

An interaction between two groups of nodes of different types can be represented as a bipartite hyperedge $h = (\{v_{1^r}, \ldots, v_{k^r}\}, \{v_{1^l}, \ldots, v_{k^l}\})$. Here, $\{v_{1^r}, \ldots, v_{k^r}\} \in \mathcal{H}^r$ is the right hyperedge with nodes from set $\mathcal{V}^r$, $\{v_{1^l}, \ldots, v_{k^l}\} \in \mathcal{H}^l$ is the left hyperedge with nodes from set $\mathcal{V}^l$, and $\mathcal{V}^r \cap \mathcal{V}^l = \emptyset$. For defining conditional intensity function, similar to homogenous hyperedges, we define $\lambda_h(t) = f(\{\mathbf{v}_{1^r}, \ldots, \mathbf{v}_{k^r}\}, \{\mathbf{v}_{1^l}, \ldots, \mathbf{v}_{k^l}\})$. Here $f(.) \geq 0$ is defined by a neural network, and $\mathbf{v}_{i^r}(t), \mathbf{v}_{i^l}(t)$ are node embeddings of the nodes in h. This involves a Cross Attention Layer (CAT) between the sets of nodes to create dynamic embeddings. This is done by finding the attention weight as shown below,

$$\begin{aligned}
\gamma_{i^r j^l} &= (\mathbf{W}_{Q^r}^T \mathbf{v}_{i^r}(t))^T \mathbf{W}_{K^l}^T \mathbf{v}_{j^l}(t), \forall 1^r \leq i^r \leq k^r, 1^l \leq j^l \leq k^l, \\
\gamma_{i^l j^r} &= (\mathbf{W}_{Q^l}^T \mathbf{v}_{i^l}(t))^T \mathbf{W}_{K^r} \mathbf{v}_{j^r}^T(t), \forall 1^l \leq i^l \leq k^l, 1^r \leq j^r \leq k^r \\
\alpha_{i^r j^l} &= \frac{\exp\left(\gamma_{i^r j^l}\right)}{\sum\limits_{1^l \leq \ell^l \leq k^l} \exp\left(\gamma_{i^r \ell^l}\right)} \qquad \alpha_{i^l j^r} = \frac{\exp\left(\gamma_{i^l j^r}\right)}{\sum\limits_{1^r \leq \ell^r \leq k^r} \exp\left(\gamma_{i^l \ell^r}\right)}.
\end{aligned} \tag{3.10}$$

Then use them to find the dynamic embeddings for left node v_{i^l} and right node v_{i^r} as follows,

$$\mathbf{d}_{v_i^r}^h = \tanh\left(\sum_{1^l \leq j^l \leq k^l} \alpha_{i^r j^l} \mathbf{W}_{V^l}^T \mathbf{v}_{j^l}(t)\right) \qquad \mathbf{d}_{v_i^l}^h = \tanh\left(\sum_{1^r \leq j^r \leq k^r} \alpha_{i^l j^r} \mathbf{W}_{V^r}^T \mathbf{v}_{j^r}(t)\right). \tag{3.11}$$

Here, $\mathbf{W}_{Q^r}, \mathbf{W}_{K^r}, \mathbf{W}_{V^r} \in \mathbb{R}^{d \times d}$ and $\mathbf{W}_{Q^l}, \mathbf{W}_{K^l}, \mathbf{W}_{V^l} \in \mathbb{R}^{d \times d}$ are the learnable weights for nodes in the left and right, respectively. Then static embeddings are created for both left $\mathbf{s}_{v_i^r}^h = \mathbf{W}_{s^r} \mathbf{v}_{i^r}(t)$ and right side $\mathbf{s}_{v_i^l}^h = \mathbf{W}_{s^l} \mathbf{v}_{i^l}(t)$, where $\mathbf{W}_{s^r}, \mathbf{W}_{s^l} \in \mathbb{R}^{d \times d}$ are learnable parameters. Then we calculate the Hadamard power of the difference between static and dynamic embedding pairs for both the left and right nodes, followed by a linear layer and average pooling to get the final score $\mathcal{P}^h$ as shown below,

$$\begin{aligned}
o_{i^r} &= \mathbf{W}_o^T (\mathbf{d}_{v_i^r}^h - \mathbf{s}_{v_i^r}^h)^2 + b_o, \qquad o_{i^l} = \mathbf{W}_{o^l}^T (\mathbf{d}_{v_i^l}^h - \mathbf{s}_{v_i^l}^h)^2 + b_{o^l}, \\
\mathcal{P}^h &= \frac{1}{k^r} \sum_{i=1^r}^{k^r} \log(1 + \exp\left(o_{i^r}\right)) + \frac{1}{k^l} \sum_{i=1^l}^{k^l} \log(1 + \exp\left(o_{i^l}\right)).
\end{aligned} \tag{3.12}$$

Datasets	$\lvert\mathcal{V}\rvert$	$\lvert\mathcal{E}(T)\rvert$	$\lvert\mathcal{H}\rvert$
email-Enron	143	10,883	1,542
email-Eu	998	234,760	25,791
congress-bills	1,718	260,851	85,082
NDC-classes	1,161	49,724	1,222
NDC-sub	5,311	112,405	10,025

Table 3.2: Datasets used for Homogeneous Hyperedges along with their vital statistics.

Here, $\mathbf{W}_{o^r}, \mathbf{W}_{o^l} \in \mathbb{R}^{d\times 1}$ and $b_{o^l}, b_{o^l} \in \mathbb{R}$ are learnable parameters of output layer. In our model the value of $f(\{\mathbf{v}_{1^r}(t), \mathbf{v}_{2^r}(t), \ldots, \mathbf{v}_{k^r}(t)\}, \{\mathbf{v}_{1^l}(t), \mathbf{v}_{2^l}(t), \ldots, \mathbf{v}_{k^l}(t)\}) = \mathcal{P}^h$.

Now to learn good dynamic representation for nodes, we follow the same architecture explained in Section 3.1.3. But, the left and right hyperedges have their own set of parameters for the *Temporal drift*, *History aggregation*, and *Interaction update* stages. Further, for dynamic embeddings in the *Interaction update* stage, nodes in the left hyperedge have it as a function of embeddings of nodes in the right hyperedge, and vice versa. This function shares its parameters with the conditional intensity function, as shown in Equation 3.11.

For learning parameters, we follow the same procedure as that of homogenous hyperedges as explained in Section 3.2, except the negative sampling is done differently. We keep the left or right hyperedge fixed for generating negative samples and add a corrupted hyperedge on the other side. For example, for generating left hyperedge negative samples, we replace the right hyperedge by selecting a random subset of nodes from the right node-set $\mathcal{V}^r$. The size of the corrupted hyperedge is selected based on the categorical distribution of sizes of the right hyperedge in the training set.

3.4 Experimental Settings

3.4.1 Datasets

All the datasets for homogeneous hyperedge interactions in Table 3.2 are taken from work (Benson et al., 2018)[1]. The following are the different categories of the datasets and their explanation.

Email Network [email-Enron (Klimt and Yang, 2004); email-Eu (Paranjape et al., 2017)]. This is a sequence of timestamped email interactions between employees at a company. The entities involved are the email addresses of the sender and receivers. The timestamps in this

[1]https://www.cs.cornell.edu/ arb/data/

Datasets	$\lvert\mathcal{V}^r\rvert$	$\lvert\mathcal{V}^l\rvert$	$\lvert\mathcal{E}(T)\rvert$	$\lvert\mathcal{H}^r\rvert$	$\lvert\mathcal{H}^l\rvert$
CastGenre	5,763	20	12,295	11,665	1,078
CastKeyword	3,998	1,953	13,826	12,365	10,737
CastCrew	5,763	4,541	12,295	11,665	9,451

Table 3.3: Datasets used for Bipartite Hyperedges along with their vital statistics.

are recorded at a resolution of 1 millisecond and are scaled down in all our experiments. The scaling factor is chosen as the median value of interevent duration for both datasets.

congress-bills (Fowler, 2006). Here, interactions are the legislative bills put forth in the House of Representatives and the Senate in the US. The entities involved in an interaction are the US congresspersons who sponsor and co-sponsor the bill. The timestamp in this is the date when the bill is introduced.

Drug Networks [NDC-classes, NDC-sub]. Each interaction is a sequence of drugs, and timestamps are the dates at which the drugs were introduced in the market. In NDC classes, nodes involved in the interaction are class labels applied to the drugs. In NDC-sub, nodes involved in the interaction are substances that make up the drug.

The bipartite hyperedge interactions datasets 3.3 are prepared from Kaggle's The Movies Dataset [1]. The following are the three datasets created from it.

[CastGenre, CastKeyword, CastCrew] For the CastGenre dataset, nodes in the left hyperedge are movie actors and the genres associated with the movie is in the right hyperedge. For the CastKeyword dataset, we use keywords associated with movie plots for the right hyperedge instead of genres. Here, the movie release date is the timestamp associated with each hyperedge. For the CastCrew dataset, we use the names of the crew associated with movie production for the right hyperedge. We only considered movies released after 1990 and removed less frequent entities for preparing these datasets.

3.4.2 Baselines

In Table 3.4, we have compared the properties of the baseline models we created against the proposed models HyperTPP for the homogenous hyperedge interactions and BHyperTPP for the bipartite hyperedge interactions. Here, models EdgeTPP-(no-HG) and EdgeTPP-(no-HGIU) use pairwise edge models instead of hyperedge interaction. Similarly, model BEdgeTPP-(no-HG) uses bipartite pairwise edges to model bipartite hyperedge interaction. Further, in models

[1] https://www.kaggle.com/datasets/rounakbanik/the-movies-dataset

Methods	T.D.	H.G.	I.U.	Hyperedge	Bipartite
HyperTPP-(no-HGIUTD)	✗	✗	✗	✓	✗
HyperTPP-(no-HGTD)	✓	✗	✓	✓	✗
EdgeTPP-(no-HGIU)	✓	✗	✗	✗	✗
EdgeTPP-(no-HG)	✓	✗	✓	✗	✗
HyperTPP-(no-HGIU)	✓	✗	✗	✓	✗
HyperTPP-(no-HG)	✓	✗	✓	✓	✗
HyperTPP-(no-IU)	✓	✓	✗	✓	✗
HyperTPP	✓	✓	✓	✓	✗
BEdgeTPP-(no-HG)	✓	✗	✓	✗	✓
BHyperTPP-(no-HG)	✓	✗	✓	✓	✓
BHyperTPP	✓	✓	✓	✓	✓

Table 3.4: Models and their properties. Here, ✓ indicates the usage of that property, and ✗ indicates the absence of that property. Here, `T.D.` corresponds to `Temporal drift`, `H.G` corresponds to `History aggregation`, and `I.U.` corresponds to `Interaction update`.

HyperTPP-(no-HGIUTD) and HyperTPP-(no-HGTD), conditional intensity is modeled as the Rayleigh process as in Equation 3.4, and duration predictions are made using the closed-form expression in Equation 3.15. HyperTPP-(no-HG) is the model that uses hyperedges for predicting interactions and has the same dynamic node presentation as of EdgeTPP-(no-HG) model. So, in our studies, we will compare these two models to claim and establish the advantage of hyperedge modeling over pairwise modeling. Similarly, HyperTPP-(no-HGIU) is the hyperedge model version of the EdgeTPP-(no-HGIU) model. A detailed description of each model is provided below.

HyperTPP-(no-HGIUTD) . In this, we keep embeddings $\mathbf{v}(t)$ fixed for every time instance. Then intensity is modeled as the Rayleigh process as in Equation 3.4, and duration predictions are made using the closed form expression in Equation 3.15.

HyperTPP-(no-HGTD). Similar to HyperTPP-(no-HGIUTD) , but here we allow the node embeddings $\mathbf{v}(t)$ to evolve when an interaction involving the node v occurs as per the *Interaction Update* stage but not through *Temporal Drift* and *History Aggregation* stage. So, $\mathbf{v}(t)$ is piecewise continuous (node embeddings are constant during interevent time) as we do not allow the model to evolve the node embedding during the interevent time. This model is based on the DeepCoevolve model used for sequential recommendation (Dai et al., 2016).

EdgeTPP-(no-HGIU). In this model, each hyperedge event is modeled as a concurrent pairwise edge event. For example, $h = \{v_1, v_2, v_3\}$ is modeled as concurrent edge events $\{(v_1, v_2), (v_2, v_3), (v_1, v_3)\}$. Here, node embeddings $\mathbf{v}(t)$ are allowed to evolve during the interevent time using *Temporal Drift* stage. For predicting interaction at time t, we use the product of all conditional intensity functions of concurrent edge events (e.g. $\lambda_{v_1,v_2}(t), \lambda_{v_2,v_3}(t), \lambda_{v_1,v_3}(t)$ for interaction h). For predicting the time of interaction, we calculate $\hat{t}$ in Equation 3.14 for each edge in the interaction and average them. In this model, conditional intensity is defined as $\lambda_{v_1(t),v_2(t)} = \mathbf{v}_1^T(t)\mathbf{v}_2(t)$.

EdgeTPP-(no-HG). This uses the same modeling technique as EdgeTPP-(no-HGIU). In addition, it also uses *Interaction Update* stage to evolve embeddings.

HyperTPP-(no-HGIU). This uses hyperedge modeling explained in Section 3.1.3, but do not use *Interaction Update* and *History Aggregation* stage.

HyperTPP-(no-HG). This uses hyperedge modeling explained in Section 3.1.3, but does not use *History Aggregation* stage.

HyperTPP-(no-IU). This uses hyperedge modeling explained in Section 3.1.3, but does not use *Interaction Update* stage.

In the case of bipartite hyperedge interactions, we will compare models BHyperTPP-(no-HG) and BEdgeTPP-(no-HG).

BEdgeTPP-(no-HG). This is the bipartite version of EdgeTPP-(no-HG), where each bipartite hyperedge is modeled as concurrent bipartite edges. For example, $h = (\{v_{1^r}, v_{2^r}\}, \{v_{1^l}, v_{2^l}\})$ is modeled as concurrent edge events $\{(v_{1^r}, v_{1^l}), (v_{2^r}, v_{1^l}), (v_{1^r}, v_{2^l}), (v_{2^r}, v_{2^l})\}$. For learning node representation, the left and right nodes have different parameters, as explained in Section 3.3.

BHyperTPP-(no-HG). This is the bipartite version of HyperTPP-(no-HG).

Apart from the models mentioned above, we also provide comparisons against the current state-of-the-art approaches for temporal pairwise graphs DyRep (Trivedi et al., 2017), TGAT (da Xu et al., 2020), and TGN (Rossi et al., 2020). In this, DyRep uses the temporal point process framework for modeling edges and can predict both type and time of interaction. The other two models, TGAT and TGN, use a continuous time dynamic graph neural network for modeling edges and can only predict the type of interaction occurring at a particular time.

DyReP. (Trivedi et al., 2019) This is designed for forecasting pairwise links in a dynamic graph. The architecture of this is similar to **EdgeTPP-(no-HG)** model, however it uses graph neural network based neighborhood aggregation in the *Interaction Update Stage* and

do not use *Temporal Drift* to evolve the embedding during the interevent time. We used our implementation of this model to get comparable performance.

TGAT. (da Xu et al., 2020) This is a dynamic link prediction model for temporal graphs. For learning dynamic node embeddings, it uses graph attention to aggregate the topological features by considering the temporal history of the data. We transferred the hypergraph dataset to the pairwise edge by clique expansion and used the code [1] provided by the authors to learn the model.

TGN. (Rossi et al., 2020) This model is for link prediction in temporal graphs. In this, interactions are stored in the form of messages and are used to update the respective nodes' memory modules. This memory embedding is used as an input to the temporal graph attention module for getting the dynamic node embedding. We followed the same preprocessing as for TGAT and used the code [2] provided by the authors to learn the model.

3.4.3 Prediction Tasks

Using temporal point process models, one can predict both the next event type and the time of the event. The following are the equivalent tasks in our settings.

Interaction type prediction. The type of event that occurs at time t can be predicted by finding the h_i with the maximum intensity value at that time, as shown below,

$$\hat{h} = \arg\max_{h_i} \lambda_{h_i}(t). \tag{3.13}$$

Interaction duration prediction. For event type h occurred at time t_h^p, to predict the duration of the future occurrence, we have to calculate the expected time t with respect to the conditional distribution $\mathrm{P}_h(t)$ in Equation 3.1,

$$\hat{t} = \int_{t_h^p}^{\infty} (t - t_h^p)\mathrm{P}_h(t)\mathrm{d}t. \tag{3.14}$$

If $\lambda(t)$ is modeled using a Rayleigh process as in Equation 3.4, we calculate the $\hat{t}$ in close form as,

$$\hat{t} = \sqrt{\frac{\pi}{2\exp f(\mathbf{v}_1(t_h^p), \mathbf{v}_2(t_h^p), \ldots, \mathbf{v}_k(t_h^p))}}. \tag{3.15}$$

[1] https://github.com/StatsDLMathsRecomSys/Inductive-representation-learning-on-temporal-graphs
[2] https://github.com/twitter-research/tgn

Otherwise, we have to compute the integration by sampling.

3.4.4 Metrics of Evaluation

Mean Reciprocal Rank (MRR). We use this for evaluating the performance of interaction prediction at time t. For finding this, we find the reciprocal of rank (r_i) of the true hyperedge against candidate negative hyperedge in descending order of $\lambda_h(t)$ and then average them for all samples in the test set, $MRR = \frac{1}{N}\sum_{i=1}^{N}\frac{1}{r_i+1}$. Here, better-performing models have higher MRR values.

Mean Absolute Error (MAE) . We use this for evaluating the performance of interaction duration prediction, $MAE = \frac{1}{N}\sum_{i=1}^{N}|\hat{t}_i - t_i^{true}|$. Here, better-performing models have lower MAE values.

3.4.5 Parameter Settings

For all experiments, we use the learning rate of 0.001, the embedding size d is fixed at 64 for homogeneous hyperedges and bipartite hyperedges, the batch size $\mathcal{B}$ is fixed as 128, and the negative sampling is fixed as $N^q = 20$, and the training is done for 100 epochs. For the Monte Carlo estimate of log of survival probability in Section 3.2.2, we use $N^t = 20$ for datasets email-Enron, email-EU, and NDC-classes, $N^t = 5$ for NDC-sub and congress-bills datasets. The choice of N^t is made by considering memory constraints. All models are implemented in PyTorch (Paszke et al., 2019), and all training is done using its Adam (Kingma and Ba, 2015) optimizer. For all datasets, we use the first 50% of interactions for training, the next 25% for validation, and the rest for testing. All the reported scores are the average of ten randomized runs along with their standard deviation.

3.5 Results

In Table 3.5, one can see that the proposed model *Hyperedge Temporal Point Process* (HyperTPP) performs better than baselines in almost all the settings. Further, it significantly outperforms HyperTPP-(no-HGIUTD), which uses static, and HyperTPP-(no-HGTD), which uses piece-wise constant node embeddings. Even though these models have a closed-form expression for the event probability and duration estimation $\hat{t}$, their performance is poor compared to HyperTPP, which has conditional intensity as a function of its dynamic node representation. We can also see that by comparing baselines, HyperTPP-(no-HG) to HyperTPP-(no-HGTD) and HyperTPP-(no-HGIUTD) to HyperTPP-(no-HGIU), the models HyperTPP-(no-HG) and HyperTPP-(no-HGIU) perform better as they use dynamic node representations as input to the conditional intensity function. This is because those models have more expressiveness and

do not assume a parametric form for $\lambda_h(t)$.

Further, one can observe the advantage of using hyperedge for modeling interactions when comparing models HyperTPP-(no-HG) to EdgeTPP-(no-HG) and HyperTPP-(no-HGIU) to EdgeTPP-(no-HGIU). This comparison is important because those models use the same dynamic node presentation, but HyperTPP-(no-HG) uses hyperedge modeling, and EdgeTPP-(no-HG) uses pairwise edge modeling. The same applies to the comparison of HyperTPP-(no-HGIU) to EdgeTPP-(no-HGIU). Between HyperTPP-(no-HG) and EdgeTPP-(no-HG), there is an improvement in the MRR metric in interaction type prediction tasks for all datasets. There is a 38.4% percentage gain in MRR for HyperTPP-(no-HG) compared to EdgeTPP-(no-HG) and a 40.3% gain for HyperTPP-(no-HGIU) compared to EdgeTPP-(no-HGIU). In the interaction duration prediction task, we can observe a significant reduction in MAE for all the datasets except for the email-Enron and email-Eu datasets. A similar observation can be made between HyperTPP-(no-HGIU) and EdgeTPP-(no-HGIU) models. This is because more than 70% of interactions are pairwise in both of those datasets. Even though pairwise edges can achieve reasonable performance, we cannot identify the hyperedge among them if there are concurrent hyperedges with common nodes, as explained in Figure 3.1. Hence, hyperedge models perform better than pairwise models for interaction type prediction.

Comparison with previous works. In Table 3.5, we have compared our model HyperTPP against previous works on pairwise temporal graphs DyRep, TGAT, and TGN. Here, we can see that our model considerably outperforms previous works as they are not hyperedge prediction models. The margin of performance is higher for datasets congress-bills and NDC-Sub compared to email-Enron and email-Eu as they have more higher-order interactions than the latter. Further, the low performance of DyRep compared to other works TGAT and TGN is because of the staleness in embedding Kazemi et al. (2020) as it does not have a temporal drift stage to model evolve embedding during the interevent stage. The poor performance of TGAT and TGN is also because they use edge and node attributes along with graph neural networks to find the dynamic embeddings of the node. This information is not available for the datasets used in these experiments, so we initialized the node embeddings with random features and did not use edge features.

Methods	email-Enron		email-Eu		congress-bills		NDC-classes		NDC-sub	
	MRR	**MAE**	**MRR**	**MAE**	**MRR**	**MAE**	**MRR**	**MAE**	**MRR**	**MAE**
DyRep[a]	42.08 ± 1.14	23.74 ± 1.68	32.23 ± 0.70	22.77 ± 0.87	40.30 ± 0.40	6.83 ± 0.12	65.80 ± 1.14	3.50 ± 0.20	55.71 ± 1.56	5.35 ± 0.43
TGAT[b]	52.60 ± 0.95	*N/A*	57.53 ± 0.89	*N/A*	78.47 ± 1.52	*N/A*	83.1 ± 0.72	*N/A*	77.06 ± 1.30	*N/A*
TGN[c]	54.32 ± 1.02	*N/A*	64.63 ± 1.2	*N/A*	80.61 ± 1.8	*N/A*	91.06 ± 0.81	*N/A*	79.33 ± 1.90	*N/A*
HyperTPP-(no-HGIUTD)	34.45 ± 0.81	127.19 ± 12.17	52.37 ± 0.47	26.43 ± 0.47	32.14 ± 0.38	54.13 ± 9.60	87.64 ± 1.63	8.98 ± 1.09	74.33 ± 0.30	4.66 ± 0.15
HyperTPP-(no-HGTD)	26.73 ± 2.76	34.22 ± 0.49	27.68 ± 5.02	17.54 ± 0.68	52.90 ± 3.24	2.44 ± 0.22	81.17 ± 2.56	6.29 ± 0.91	66.46 ± 1.50	2.52 ± 0.07
EdgeTPP-(no-HGIU)	30.84 ± 0.29	48.50 ± 0.65	43.47 ± 1.69	21.21 ± 0.06	43.27 ± 0.23	4.07 ± 0.32	60.64 ± 0.19	5.02 ± 0.15	64.90 ± 0.29	12.38 ± 0.84
EdgeTPP-(no-HG)	52.89 ± 0.38	16.52 ± 3.14	44.27 ± 0.74	19.37 ± 1.27	56.27 ± 2.89	2.65 ± 0.42	64.58 ± 0.78	3.60 ± 0.38	65.83 ± 1.41	13.82 ± 1.93
HyperTPP-(no-HGIU)	50.25 ± 1.65	88.14 ± 4.59	58.38 ± 0.20	33.56 ± 0.82	84.50 ± 0.13	3.84 ± 0.26	88.68 ± 0.71	1.92 ± 0.12	79.31 ± 0.41	3.03 ± 0.01
HyperTPP-(no-HG)	60.57 ± 1.97	25.48 ± 4.37	64.03 ± 2.22	19.72 ± 2.00	$\mathbf{92.21 \pm 0.19}$	1.87 ± 0.22	88.93 ± 0.16	1.84 ± 0.21	86.52 ± 0.18	3.49 ± 0.14
HyperTPP-(no-IU)	$\mathbf{65.26 \pm 1.24}$	18.25 ± 0.43	60.69 ± 0.09	24.66 ± 0.77	85.31 ± 0.10	3.44 ± 0.34	$\mathbf{91.24 \pm 0.74}$	1.42 ± 0.08	80.73 ± 0.15	1.75 ± 0.19
HyperTPP	62.21 ± 2.85	$\mathbf{16.12 \pm 1.45}$	$\mathbf{66.12 \pm 2.90}$	$\mathbf{15.18 \pm 2.14}$	92.09 ± 0.03	$\mathbf{1.65 \pm 0.06}$	91.01 ± 0.35	$\mathbf{1.21 \pm 0.04}$	$\mathbf{86.92 \pm 0.51}$	$\mathbf{1.65 \pm 0.18}$

Table 3.5: Performance of dynamic homogeneous hyperedge forecasting in tasks of interaction type and interaction duration prediction. Here, interaction type prediction is evaluated using MRR in %, and interaction duration prediction is evaluated using MAE. The proposed model HyperTPP beats baseline models in almost all the settings. **Citations**: [a] Trivedi et al. (2019), [b] da Xu et al. (2020), [c] Rossi et al. (2020)

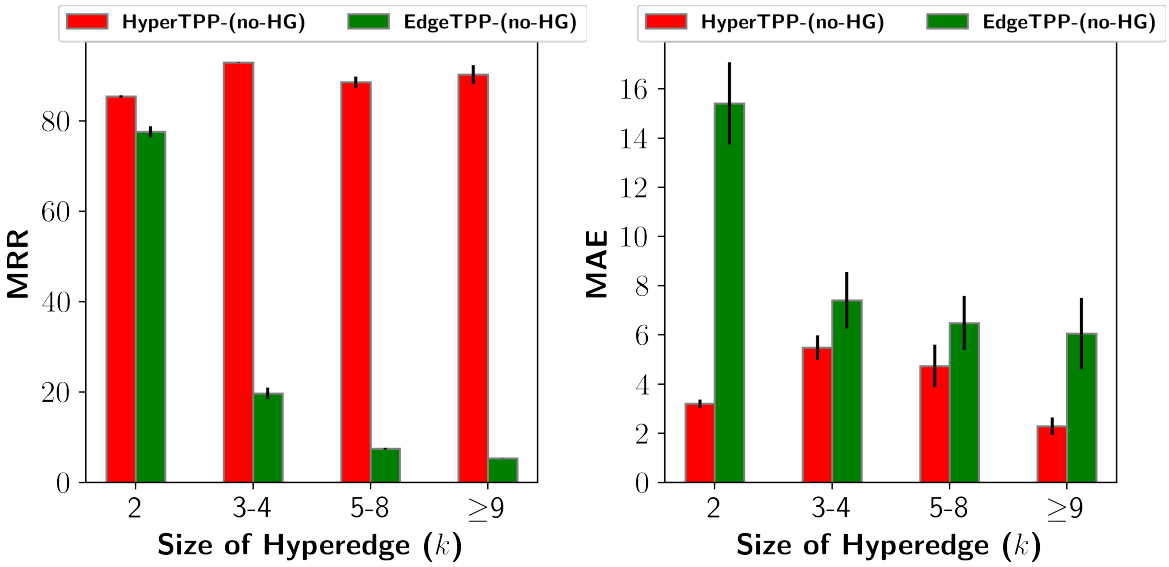
Methods	CastGenre		CastKeyword		CastCrew	
	MRR	**MAE**	**MRR**	**MAE**	**MRR**	**MAE**
BEdgeTPP-(no-HG)	23.55 ± 0.75	10.34 ± 0.34	13.61 ± 0.14	21.98 ± 0.60	13.61 ± 0.24	22.23 ± 1.03
HyperTPP-(no-HG)	27.58 ± 0.88	$\mathbf{2.88 \pm 0.43}$	36.18 ± 1.34	15.32 ± 1.75	26.03 ± 1.75	9.29 ± 2.01
BHyperTPP-(no-HG)	33.22 ± 0.52	2.91 ± 0.26	38.77 ± 1.69	9.61 ± 2.33	37.29 ± 2.65	9.18 ± 1.37
HyperTPP	27.59 ± 1.60	19.39 ± 1.62	35.32 ± 1.96	22.64 ± 1.50	25.19 ± 2.83	8.85 ± 2.07
BHyperTPP	$\mathbf{33.65 \pm 1.58}$	4.16 ± 0.84	$\mathbf{41.32 \pm 1.74}$	$\mathbf{9.27 \pm 1.67}$	$\mathbf{42.77 \pm 2.00}$	$\mathbf{8.77 \pm 1.68}$

Table 3.6: Performance of dynamic bipartite hyperedge forecasting in tasks of interaction type and interaction duration prediction. Proposed model BHyperTPP beats baseline models in almost in all the settings.

Performance on Bipartite interactions. In Table 3.6, one can observe that the proposed model *Bipartite Hyperedge Temporal Point Process* (BHyperTPP) performs better than all the other baselines in almost all settings. We can see that the models that use the bipartite property of the interaction perform better than their homogeneous counterparts. This can be inferred by the better performance of BHyperTPP-(no-HG) compared to HyperTPP-(no-HG) and BHyperTPP compared to HyperTPP. There is a gain of 23.6% in MRR and a reduction of 12.4% MAE for BHyperTPP-(no-HG) compared to HyperTPP-(no-HG). Similarly, there is a gain of 36.2% in MRR and a reduction of 46.1% MAE for BHyperTPP compared to HyperTPP. This is because these models are more expressive and consider the bipartite nature of the interaction, as explained in Section 3.3. Similar to the case of homogeneous interactions, the pairwise model performs considerably poorer than the bipartite hyperedge models. It can be inferred from the poor performance of the BEdgeTPP-(no-HG) model, which uses a bipartite pairwise edge, compared to BHyperTPP-(no-HG), which uses bipartite hyperedge for interaction modeling. Hence, one can conclude that bipartite hyperedge modeling represents data more accurately than pairwise modeling.

3.5.1 Ablation Studies

Effect of interaction update on performance. Here, we compare the performance of models that have this particular state in their update equation vs the models that do not. Firstly, we can observe a reduction of MAE by 24.7% and a gain of 3.9% in MRR for HyperTPP compared to HyperTPP-(no-IU). Similarly, HyperTPP-(no-HG) outperforms HyperTPP-(no-HGIU) considerably in the interaction type prediction task for all datasets. One can also observe a significant reduction in MAE for email-Enron and email-Eu datasets. Between HyperTPP-(no-HGIUTD) and HyperTPP-(no-HGTD), one can see that HyperTPP-(no-HGTD) has a



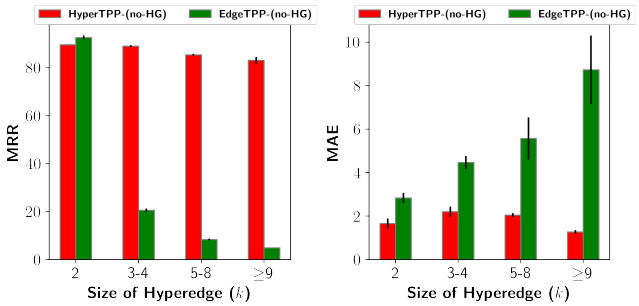
(a) (b)

Figure 3.2: Figure 3.2a shows the effect of hyperedge size (k) on interaction type prediction, and Figure 3.2b shows the effect on interaction duration prediction for NDC-sub dataset

performance gain in interaction duration prediction in all datasets. Even though HyperTPP-(no-HGIUTD) performs better in MRR in most datasets, one can see MRR for HyperTPP-(no-HGTD) is much better in congress-bills datasets. Hence one can conclude that using `Interaction update` stage resulted in performance improvement.

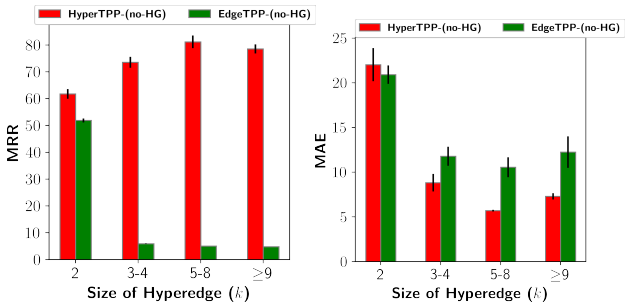
Effect of temporal drift on performance. Similar to the earlier study, the advantage of this stage can be observed by comparing the performance of HyperTPP-(no-HGIUTD) to HyperTPP-(no-HGIU) and HyperTPP-(no-HGTD) to HyperTPP-(no-HG). There is an average of 44% gain in MRR and 42.03% reduction in MAE for HyperTPP-(no-HGIU) compared to HyperTPP-(no-HGIUTD) . Further, HyperTPP-(no-HG) uniformly outperforms HyperTPP-(no-HGTD) for interaction type prediction in all datasets. There is an average of 90% improvement in the MRR for the interaction type prediction task and a 13.7% decrease in the MAE error for the duration prediction task. Hence, `Temporal drift` stage helps in performance improvement.

Effect of history aggregation on performance. The advantage of this stage can be observed by comparing HyperTPP to HyperTPP-(no-HG) and HyperTPP-(no-IU) to HyperTPP-(no-HGIU). We can see HyperTPP outperforms HyperTPP-(no-HG) in all the tasks except in Congress bills. There is an average gain of 1.7% in MRR and a 31.75% reduction in MAE for HyperTPP compared to HyperTPP-(no-HG). We can see HyperTPP-(no-IU) outperforms



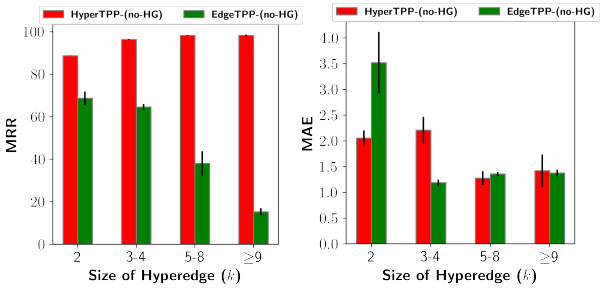
(a) (b)

Figure 3.3: Figure 3.3a shows the effect of hyperedge size (k) on interaction type prediction, and Figure 3.3b shows the effect on interaction duration prediction for NDC-classes dataset



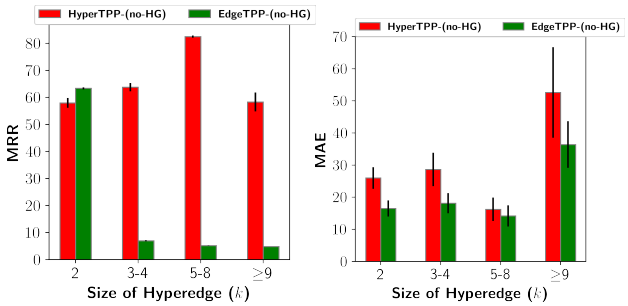
(a) (b)

Figure 3.4: Figure 3.4a shows the effect of hyperedge size (k) on interaction type prediction, and Figure 3.4b shows the effect on interaction duration prediction for email-Eu dataset



(a) (b)

Figure 3.5: Figure 3.5a shows the effect of hyperedge size (k) on interaction type prediction, and Figure 3.5b shows the effect on interaction duration prediction for congress-bills dataset



(a) (b)

Figure 3.6: Figure 3.6a shows the effect of hyperedge size (k) on interaction type prediction, and Figure 3.6b shows the effect on interaction duration prediction for Enron dataset

HyperTPP-(no-HGIU) in all settings. A similar observation can be made in bipartite datasets when comparing models BHyperTPP and BHyperTPP-(no-HG). Both models give a comparable performance for interaction duration prediction except for CastGenre dataset. For interaction type prediction BHyperTPP achieves a gain of 7% in MRR when compared to BHyperTPP-(no-HG).

3.5.2 Hyperedge Size (k) Vs Performance

While both the models HyperTPP-(no-HG) and EdgeTPP-(no-HG) use the same dynamic representations, they have different conditional intensity functions. While HyperTPP-(no-HG) uses a hyperedge-based conditional intensity function, and EdgeTPP-(no-HG) uses a pairwise

edge-based function. To compare them, we divide hyperedges into different groups based on their sizes, and the mean value of the evaluation metric is calculated for each group. These groups are defined so that each group has enough samples to make the comparisons statistically significant. In Figures 3.3a and 3.3b, we have shown the effect of hyperedge size on our model HyperTPP-(no-HG), which uses hyperedge modeling, and compared it against model EdgeTPP-(no-HG), which uses pairwise edges, on NDC-sub dataset. Here, each hyperedge is grouped into groups with $k = 2$, $3 \leq k \leq 4$, $5 \leq k \leq 8$ and $k \geq 9$. From Figure 3.3a, we can see that performance of EdgeTPP-(no-HG) is poor for hyperedges with a size of more than 2, but our model HyperTPP-(no-HG) has almost similar performance for hyperedges of different sizes. The reason for this is that HyperTPP-(no-HG) is suited more for hyperedges of varying sizes compared to EdgeTPP-(no-HG). We observed a similar trend in other datasets, email-Eu, congress-bills, NDC-classes, and Enron are shown in Figures 3.4, 3.5, 3.6, and 3.2, respectively. For the interaction duration prediction task, the error for the EdgeTPP-(no-HG) model increases with hyperedge size, while the HyperTPP-(no-HG) model performs similarly for all hyperedge sizes.

3.6 Summary

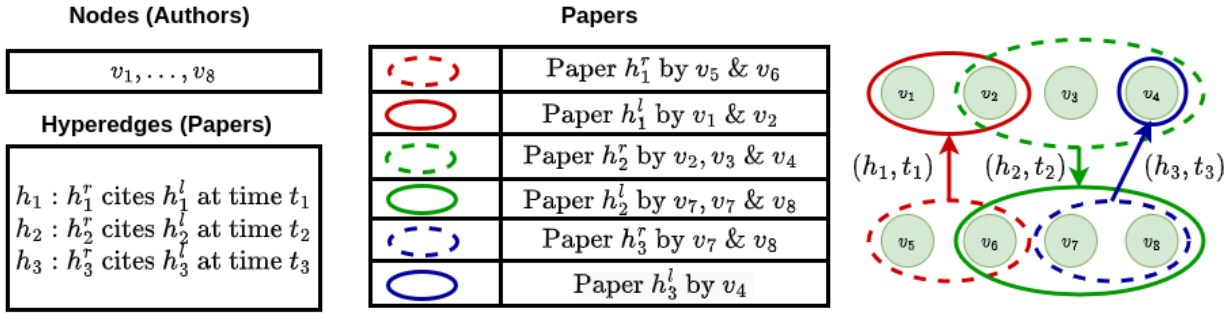
The problem of forecasting events in time-evolving hypergraphs is very challenging, and this has not been studied before. In this work, we provide the first solution to this problem by proposing a model for forecasting interaction between nodes in a network as temporal hyperedge formation events. For this, we develop a mechanism that uses the temporal point process to learn the dynamic representation of nodes. Our future work includes extending the proposed model for modeling multi-relational higher orders interactions (Fatemi et al., 2020) and incorporating multi-hop information into node representation using hypergraph neural network-based techniques for better predictive performance. We also plan to explore options to reduce the training time by using t-batch (Kumar et al., 2019) based approach that can use parallel training techniques.

Chapter 4

DHyperNodeTPP for Events in Directed HyperGraph

In this chapter, we are modeling directed interaction between a group of homogeneous entities. Examples of these types of interactions can be seen in politics, where parties form coalitions to achieve a parliamentary majority. Similarly, in citation networks, authors draw upon the works of other authors to support their arguments or to build upon existing knowledge. In this work, we provide a solution to this problem as events in a directed hypergraph. An example of events in a citation network as directed hyperedges formation is depicted in Figure 4.1. The ability to predict these events have important applications in domains such as blockchain (Ranshous et al., 2017) and recommendation systems (Liu et al., 2018).

Our solution to the abovementioned problem involves two components: (**i**) a directed hyperedge link predictor and (**ii**) a dynamic node embedding architecture. Previous works use hypergraph neural network based unsupervised learning to predict hyperlinks (Wei et al., 2022, Lee and Shin, 2023a). These models cannot be applied to the proposed problem as hypergraphs cannot be constructed from the events as they exist only for an instant of time. Alternately, set prediction-based scoring functions are used for hyperedge prediction. These models use a deep learning-based permutation invariant architecture to make predictions. The current works are developed for undirected hyperedges (Zhang et al., 2019) and bipartite hyperedges (refer to Section 3.3), which only show a single type of relations, self-connection in the case of the undirected and cross connection in the case of bipartite. Unlike these, a directed hyperedge can represent three types of relations, two self-connections among the left and right groups and a cross-connection between these groups. For learning dynamic node embedding, the majority of the works are based on temporal graph neural networks that predict pair-wise edges (S. Gupta



(a) Temporal citation dataset

(b) Model

Figure 4.1: A temporal citation network modeled as a temporal directed hypergraph with 8 nodes and 3 hyperedges. Here, $t_3 > t_2 > t_1$, and hyperedge is represented as a tuple $h_i = (h_i^r, h_i^l)$ with h_i^r and h_i^l as right and left hyperedges, respectively.

and Dukkipati, 2019, Trivedi et al., 2019, Rossi et al., 2020). In Chapter 3, this architecture has been extended to undirected and bipartite hyperedges to incorporate interaction dynamics. These works achieve this using a sequential recurrent neural network-based model that updates node embedding when involved in an event. One of the main disadvantages of this approach is that this will not allow the model to do batch processing of the data, as each sample depends on the previous samples. Further, for simulating the future in directed temporal graphs with nodes $\mathcal{V}$, one needs to compare all the ${}^{|\mathcal{V}|}\mathsf{P}_2$ pairwise combinations of nodes in the network. In the case of directed hypergraphs, there are a maximum of $2^{|\mathcal{V}|}$ combinations of nodes in the left and the right hyperedges.

In this chapter, we solve the above-mentioned problems by proposing a scalable hyperedge forecasting model called *Directed HyperNode Temporal Point Process*, **DHyperNodeTPP**. Unlike previous models that forecast future events by an exhaustive search, we use the temporal point process (TPP) based generative model to forecast candidate hyperedges. This is achieved by forecasting event times for nodes followed by forecasting the projected adjacency matrix along with the distribution of hyperedge sizes of the nodes that are then used for generating candidate hyperedges. Further, to process batches of data, we use a temporal message-passing framework to store the features of interactions for each node and then use them to update the node embeddings before the next batch. The content in this chapter is also available in another form in our work (Gracious et al., 2023). The following are the main contributions of this chapter.

- A temporal point process model for forecasting directed hyperedges in a scalable way.
- A scalable dynamic representation learning approach that uses temporal graph attention

networks and batch processing.

- A directed hyperedge prediction model.
- Creation of five real-world temporal directed hypergraph datasets from open-source data.
- Extensive experiments to show the advantage of our model on directed hyperedge formation event forecasting over existing modeling techniques.

4.1 Problem Definition

Directed Hypergraph. A directed hypergraph is denoted by tuple $\mathcal{G} = (\mathcal{V}, \mathcal{H})$. Here, $\mathcal{V} = \{v_1, v_2, \ldots, v_{|\mathcal{V}|}\}$ is the set of nodes, $\mathcal{H}$ is the set of valid hyperedges. In this, each hyperedge $h = (h^r, h^l) \in \mathcal{H}$ is represented by two subset of nodes, $h^r \subset \mathcal{V}$ the right hyperedge of size $k^r = |h^r|$, and $h^l \subset \mathcal{V}$ the left hyperedge of size $k^l = |h^l|$. The maximum size of right and left hyperedge is denoted by $k^r_{max} = \max_{h\in\mathcal{H}} |h^r|$ and $k^l_{max} = \max_{h\in\mathcal{H}} |h^l|$, respectively. Table 4.1 provides the definitions for the notations used in this chapter.

Temporal Events. The sequence of events occurring in a hypergraph till time $t \in \mathbb{R}^+$ is denoted by $\mathcal{E}(t) = \{(e_1, t_1), \ldots, (e_n, t_n), \ldots\}$. Here, $e_n = \{h_{n,m}\}_{m=1}^{L_n}, h_{n,m} \in \mathcal{H}$ denotes the L_n concurrent hyperedges occurring at time $t_n \leq t$ with n as the index of the event and m as the index of the concurrent hyperedge. Then for each e_n, we create its projected adjacency matrices $\mathbf{A}^r_n, \mathbf{A}^l_n \in \{0,1\}^{|\mathcal{V}|\times|\mathcal{V}|}$ indicating the pairwise adjacency matrices for nodes in right hyperedge. In these, $\mathbf{A}^r_n[i,j] = 1$ if $\exists\ \{v_i, v_j\} \subset h^r, h = (h^r, h^l) \in e_n$ else 0, and $\mathbf{A}^l_n[i,j] = 1$ if $\exists\ v_i \in h^r$ and $v_j \in h^l, h = (h^r, h^l) \in e_n$ else 0. The rows of $\mathbf{A}^r_n$ and $\mathbf{A}^l_n$ denoted by $\mathbf{a}^r_{n,i}$ and $\mathbf{a}^l_{n,i}$, respectively, are the adjacency vectors of node v_i. In addition to this, we create the size matrices of right $\mathbf{K}^r_n \in \{0,1\}^{\mathcal{V}\times k^l_{max}}$ and left hyperedges $\mathbf{K}^l_n \in \{0,1\}^{|\mathcal{V}|\times k^r_{max}}$ with respect to the nodes in right hyperedge. Here, $\mathbf{K}^r_n[i,k] = 1$ if $\exists k = |h^r|$ and $\mathbf{K}^l_n[i,k] = 1$ if $\exists k = |h^l|\ \forall h \in e_{n+1}$ and $v_i \in h^r$. The size vectors for a node v_i is denoted by $\mathbf{k}^l_{i,n}$ and $\mathbf{k}^r_{i,n}$. These are the ith rows of respective matrices.

Goal. The model learn the probability distribution $\mathrm{P}^*(e_{n+1}, t_{n+1}) = \mathrm{P}((e_{n+1}, t_{n+1})|\mathcal{E}(t_n))$ over the time (t_{n+1}) and type (e_{n+1}) of the next event given the history, $\mathcal{E}(t_n)$. A naive implementation of this has the following likelihood,

$$\mathrm{P}^*(e_{n+1}, t_{n+1}) = \mathrm{P}^*(t_{n+1}) \prod_{h\in\mathcal{H}} [\mathrm{P}^*(h|t_{n+1})]^{\mathbb{I}_{h\in e_{n+1}}} [1 - \mathrm{P}^*(h|t_{n+1})]^{\mathbb{I}_{h\notin e_{n+1}}}.$$

This is obtained by assuming all the interactions inside event e_{n+1} are independent, given the time of the event and history. The main challenge in generating samples based on the above

Notations	Definitions
${}^{n}\mathsf{P}_{r}$	Permutations
k^r, k^l	Sizes of left and right hyperedges
L_n	Concurrent interactions occurring at time t_n
$\mathbf{V}(t) \in \mathbb{R}^{\mathcal{V} \times d}$	Node embeddings
m	Concurrent hyperedge index
n	Index of the event
k	Combined Hyperedge size ($k = k^r + k^l$)
$\mathrm{P}^*()$	Probability ignoring the conditional
$\mathrm{P}_i^*()$, $\mathrm{S}_i^*()$	Probability and Survival function for node v_i
$\mu_i(t_{n+1})$	Parameters of the time modeling for node v_i
$\mathrm{MLP}_t(.)$	MLP for time modeling
s_t	Variance of LogNormal distribution
$\mathrm{MLP}_{ar}(.), \mathrm{MLP}_{al}(.)$	MLP for adjacency modeling
$\mathrm{MLP}_{sr}(.), \mathrm{MLP}_{sl}(.)$	MLP for size modeling
N	Total number of events in the dataset
e_n	L_n Concurrent events at time t_n
L_n	Number of concurrent events in e_n
$\Phi(.)$	Cumulative density function of standard normal
i	Index of node
$\mathbf{A}_n^r, \mathbf{A}_n^l$	Right and left projected adjacency matrix for hyperedges in e_n
$\mathbf{K}_n^r, \mathbf{K}_n^l$	Right and left size vectors for hyperedges in e_n
$\boldsymbol{\kappa}_{n,i}^r / \boldsymbol{\kappa}_{n,i}^l$	Parameters of right /left size distribution at time step t_n
$\boldsymbol{\theta}_{n,i}^r / \boldsymbol{\theta}_{n,i}^r$	Parameters of right/left adjacency matrices at time step t_n
$\mathcal{LL}$	Complete negative log-likelihood
$\mathcal{H}_n^c$	Candidate hyperedges at time t_n
$\lambda_h(t)$	Hyperedge link predictor
t_i^p	Previous interaction time for node v_i
$\mathbf{d}_{i^{r/l}}^h, \mathbf{s}_{i^{r/l}}^h$	Dynamic/Static node embedding for node v_i in hyperedge $h^{r/l}$
$\mathbf{Mem}_{v_i}$	Memory embeddings for node v_i
$\mathbf{Mem}$	Memory Module
$\mathcal{B}$	Batch size
$\boldsymbol{\psi}(t)$	Time embedding function
$\{\omega_i\}_{i=1}^d$ and $\{\phi_i\}_{i=1}^d$	Learnable parameters of $\boldsymbol{\psi}(t)$
$\mathbf{msg}_i^r(t_b) / \mathbf{msg}_i^l(t_b)$	Messages vector created for node v_i in the right/left hyperedge at time t_b
$\mathcal{N}$	Hyperparameter for the temporal embeddings
$\mathcal{N}_{h^r}(t), \mathcal{N}_{h^l}(t)$	Recent $\mathcal{N}$ interactions involving node v_i in the right/left hyperedge
MultiHeadAttention	Multi-head attention
$\mathbf{q}(t)$	Query embeddings
$\mathbf{C}(t)$	Key embeddings
$\mathbf{C}(t)$	Value embeddings
$\hat{t}_{n,i}$	Next event time estimate for nth event
$\hat{\mathbf{a}}_{n,i}^r, \hat{\mathbf{a}}_{n,i}^l$	Projected adjacency vector estimates for nth event for node v_i at time t_n

Table 4.1: Notations used in this chapter

likelihood is that the hyperedge prediction requires a search over a huge number of candidates, $|\mathcal{H}|$. Alternately, we can further reduce this by introducing a candidate generation model that will forecast likely hyperedges,

$$\mathrm{P}^*(e_{n+1}, t_{n+1}) = \mathrm{P}^*(\cup e^r_{n+1}, t_{n+1})\mathrm{P}^*(\mathcal{H}^c | \cup e^r_{n+1}, t_{n+1}) \prod_{h \in \mathcal{H}^c_{n+1}} [\mathrm{P}^*(h|t_{n+1})]^{\mathbb{I}_{h \in e_{n+1}}} [1 - \mathrm{P}^*(h|t_{n+1})]^{\mathbb{I}_{h \notin e_{n+1}}}.$$

Here, $\cup e^r_{n+1} = \cup_{m=1}^{L_n} h^r_{n+1,m}$ be the set of all the nodes in the right hyperedge. In this model, we first predict all the right nodes that are observing events at time t_{n+1}, followed by a candidate hyperedge generation module that outputs $\mathcal{H}^c$, and finally, we use the directed hyperedge link predictor to get the ground truth. Here, we assume that the right hyperedge represents the source of the interactions; for email exchange, the right node is the sender, and citation networks have the right nodes as the paper's authors. The entire block diagram for the proposed framework is shown in Figure 4.3.

4.2 Model

We follow a multi-task learning framework by dividing event forecasting into three different modules that model time forecasting, candidate generation, and hyperedge prediction. These models use the dynamic embeddings of nodes $\mathbf{V}(t) \in \mathbb{R}^{|\mathcal{V}| \times d}$ as input. Here d is the dimension of embeddings, and $\mathbf{v}_i(t) \in \mathbb{R}^d$ denotes the embedding of node v_i. The architecture used to learn this is explained in Section 4.2.3. In the following section, MLP_*s are multilayer perceptron functions, and the architecture for these are explained in Section 4.2.1.

Node Event Model. Given the history, $\mathcal{E}(t_n)$, the task is to model the probability distribution of the time of occurrence of events on nodes. This is the product of likelihood of next event occurring at time $t_{n+1} = \Delta t_{n+1} + t_n$ for nodes in $\cup e^r_{n+1}$ and event not occurring for nodes not in $\cup e^r_{n+1}$ for the interval $[t_n, t_{n+1}]$,

$$\mathrm{P}^*(\cup e^r_{n+1}, t_{n+1}) = \prod_{v_i \in \cup e^r_{n+1}} \mathrm{P}^*_i(\Delta t_{n+1}) \prod_{v_i \notin \cup e^r_{n+1}} \mathrm{S}^*_i(\Delta t_{n+1}).$$

Here, $\mathrm{S}^*_i(\cdot)$ is the survival function that models the probability of an event not occurring for node v_i in the interval $[t_n, t_{n+1}]$, and $\mathrm{P}^*_i(\cdot)$ is the probability that event is observed for node v_i at time t_{n+1}. We use Lognormal distribution to model event times,

$$\Delta t_{n+1} \sim \mathrm{Lognormal}(\mu_{n+1,i}, s^2_t).$$

Here, variance s_t is a hyperparameter and $\mu_{n+1,i}$ is parameterized by a neural network that takes node embedding of v_i at time t_n as the input, $\mu_{n+1,i} = \text{MLP}_t(\mathbf{v}_i(t_n))$. Then the log-likelihood becomes,

$$\mathcal{LL}_t^{n+1} = \sum_{v_i \in \cup e_{n+1}^r} \frac{(\log(\Delta t_{n+1}) - \mu_{n+1,i})^2}{2s_t^2} - \sum_{v_i \notin \cup e_{n+1}^r} \log\left(1 - \Phi\left(\frac{\log \Delta t_{n+1} - \mu_{n+1,i}}{s_t}\right)\right). \tag{4.1}$$

Here, $\Phi(.)$ is the cumulative density function of the standard normal. The second loss component due to the survival function is approximated using negative sampling during training.

Candidate Generation. This is achieved by predicting the projected adjacency, $\mathbf{a}_{n+1,i}^r, \mathbf{a}_{n+1,i}^l$, and size vectors, $\mathbf{k}_{n+1,i}^r, \mathbf{k}_{n+1,i}^l$, for node v_i observing event from previous module, $v_i \in \cup e_{n+1}^r$. The likelihood for this stage is,

$$\text{P}^*(\mathcal{H}_{n+1}^c | \cup e_{n+1}^r, t_{n+1}) = \prod_{v_i \in \cup e_{n+1}^r} \text{P}_i^*(\mathbf{k}_{n+1,i}^r | t_{n+1}) \text{P}_i^*(\mathbf{k}_{n+1,i}^l | t_{n+1}) \text{P}_i^*(\mathbf{a}_{n+1,i}^r | t_{n+1}) \text{P}_i^*(\mathbf{a}_{n+1,i}^l | t_{n+1}).$$

Here, we assume independence of the size and adjacency matrices conditioned on the event $(\cup e_{n+1}^r, t)$, and history. We use independent Bernoulli distribution to model the adjacency vectors of v_i,

$$\mathbf{a}_{n+1,i}^r \sim \text{Bernoulli}(\sigma(\boldsymbol{\theta}_{n+1,i}^r)),$$
$$\mathbf{a}_{n+1,i}^l \sim \text{Bernoulli}(\sigma(\boldsymbol{\theta}_{n+1,i}^l)).$$

Here, $\boldsymbol{\theta}_{n+1,i}^r = \text{MLP}_{ar}(\mathbf{v}_i(t_n))$ and $\boldsymbol{\theta}_{n+1,i}^l = \text{MLP}_{al}(\mathbf{v}_i(t_n))$. Then we can write log-likelihood for adjacency prediction as,

$$\mathcal{LL}_a^{n+1} = \sum_{v_i \in \cup e_{n+1}^r, s=r,l} \langle \mathbf{a}_{n+1,i}^s, \log(\sigma(\boldsymbol{\theta}_{n+1,i}^s) \rangle + \langle (\mathbf{1} - \mathbf{a}_{n+1,i}^s), \log(\mathbf{1} - \sigma(\boldsymbol{\theta}_{n+1,i}^s)) \rangle. \tag{4.2}$$

Similarly, we model the size distribution for node v_i,

$$\mathbf{k}_{n+1,i}^r \sim \text{Bernoulli}(\sigma(\boldsymbol{\kappa}_{n+1,i}^r)),$$
$$\mathbf{k}_{n+1,i}^l \sim \text{Bernoulli}(\sigma(\boldsymbol{\kappa}_{n+1,i}^l)).$$

Here, $\boldsymbol{\kappa}^r_{n+1,i} = \text{MLP}_{sr}(\mathbf{v}_i(t_n))$, and $\boldsymbol{\kappa}^l_{n+1,i} = \text{MLP}_{sl}(\mathbf{v}_i(t_n))$. Then we can write log-likelihood for size as,

$$\mathcal{LL}^{n+1}_k = \sum_{v_i \in \cup e^r_{n+1}, s=r,l} \langle \mathbf{k}^s_{n+1,i}, \log(\sigma(\boldsymbol{\kappa}^s_{n+1,i})) \rangle + \langle (\mathbf{1} - \mathbf{k}^s_{n+1,i}), \log(\mathbf{1} - \sigma(\boldsymbol{\kappa}^s_{n+1,i})) \rangle, \tag{4.3}$$

Hyperedge Predictor. Given the candidate hyperedges $\mathcal{H}^c_{n+1}$, the the probability of observing e_{n+1} at time t_{n+1} is,

$$\text{P}^*(e_{n+1} | \mathcal{H}^c_{n+1}, t_{n+1}) = \prod_{h \in \mathcal{H}^c_{n+1}} \sigma(\lambda_h(t_{n+1}))^{\mathbb{I}_{h \in e_{n+1}}} (1 - \sigma(\lambda_h(t_{n+1})))^{\mathbb{I}_{h \notin e_{n+1}}}.$$

Here, $\lambda_h(t)$ is parameterized by a neural network that takes embeddings of the nodes in h at time t as input,

$$\lambda_h(t) = f(\{\mathbf{v}_i(t)\}_{v_i \in h^r}, \{\mathbf{v}_i(t)\}_{v_i \in h^l}).$$

The architecture of $f(.)$ is explained in Section 4.2.2. The log-likelihood is,

$$\mathcal{LL}^{n+1}_h = \sum_{h \in \mathcal{H}^c_{n+1}} \mathbb{I}_{h \in e_{n+1}} \log \sigma(\lambda_h(t_{n+1})) + \mathbb{I}_{h \notin e_{n+1}} \log(1 - \sigma(\lambda_h(t_{n+1}))). \tag{4.4}$$

While training, the candidate hyperedges are the combination of the true hyperedges and negative hyperedges generated by negative sampling. This is done by corrupting either the left or right of true hyperedge with a hyperedge of randomly sampled size and filled with random nodes.

Loss Function. The complete likelihood for event sequence $\mathcal{E}(t_N)$ is as follows,

$$\text{P}(\mathcal{E}(t_N)) = \prod_{n=0}^{N-1} \text{P}^*(e_{n+1}, t_{n+1}).$$

For training, we minimize the negative log-likelihood,

$$\mathcal{NLL} = -\sum_{n=0}^{N-1} \left(\mathcal{LL}^{n+1}_t + \mathcal{LL}^{n+1}_k + \mathcal{LL}^{n+1}_a + \mathcal{LL}^{n+1}_h \right).$$

4.2.1 MLP Layers

- $\text{MLP}_t(\cdot)$ parameterizes the mean of the LogNormal distribution in the event modeling task using the dynamic node embedding $\mathbf{v}_i(t_n)$ as shown below,

$$\mu_{n+1,i} = \text{MLP}_t(\mathbf{v}_i(t_n)) = \mathbf{W}_{t1}\tanh(\mathbf{W}_{t0}\mathbf{v}_i(t_n) + \mathbf{b}_{t0}) + b_{t1}. \tag{4.5}$$

  Here, $\mathbf{W}_{t1} \in \mathbb{R}^{1\times d}, \mathbf{W}_{t0} \in \mathbb{R}^{d\times d}, b_{t1} \in \mathbb{R}, \mathbf{b}_{t0} \in \mathbb{R}^d$.

- $\text{MLP}_{sr}(\cdot)$ parameterizes the Bernoulli distributions of the sizes of right hyperedges for the nodes in right hyperedges as shown below,

$$\boldsymbol{\kappa}^r_{n+1,i} = \text{MLP}_{sr}(\mathbf{v}_i(t_n)) = \mathbf{W}_{sr1}\tanh(\mathbf{W}_{sr0}\mathbf{v}_i(t_n) + \mathbf{b}_{sr0}) + \mathbf{b}_{sr1}, \forall\, v_i \in \cup e^r_n. \tag{4.6}$$

  Here, $\mathbf{W}_{sr1} \in \mathbb{R}^{k^r_{max}\times d}, \mathbf{W}_{sr0} \in \mathbb{R}^{d\times d}, \mathbf{b}_{sr1} \in \mathbb{R}^{k^r_{max}}, \mathbf{b}_{sr0} \in \mathbb{R}^d$. Similarly, we model $\text{MLP}_{sl}(\cdot)$ for the sizes of left hyperedges with respect to nodes in the right.

- $\text{MLP}_{ar}(\cdot)$ parameterizes the Bernoulli distributions of the projected adjacency vectors for the nodes in the right hyperedge as shown below,

$$\boldsymbol{\theta}^r_{n+1,i} = \text{MLP}_{ar}(\mathbf{v}_i(t_n)) = \mathbf{Mem}\,\tanh(\mathbf{W}_{ar0}\mathbf{v}_i(t_n) + \mathbf{b}_{ar0}), \forall\, v_i \in \cup e^r_n. \tag{4.7}$$

  Here, $\mathbf{Mem} \in \mathbb{R}^{|\mathcal{V}|\times d}, \mathbf{W}_{ar0} \in \mathbb{R}^{d\times d}, \mathbf{b}_{ar0} \in \mathbb{R}^d$. The final layer is parameterized by the memory embedding defined in Section 4.2.3.1 to capture the evolution of nodes with each interaction. Similarly, we model $\text{MLP}_{al}(\cdot)$ to parameterize the distribution of left projected adjacency vectors.

4.2.2 Architecture of Hyperedge Predictor

We now propose an architecture that utilizes all three types of relations in a directed hyperedge to predict true hyperedges from candidate hyperedges. Given a directed hyperedge $h = (\{v_{1^r}, \ldots, v_{k^r}\}, \{v_{1^l}, \ldots, v_{k^l}\})$, we use the cross-attention layer (CAT) (refer 3.3) between the sets of nodes to create cross dynamic hyperedge embeddings for node v_{i^r} as follows,

$$\mathbf{d}^{ch}_{i^r} = \text{CAT}(\{\mathbf{v}_{1^r}(t), \ldots, \mathbf{v}_{k^r}(t)\}, \{\mathbf{v}_{1^l}(t), \ldots, \mathbf{v}_{k^l}(t)\}).$$

These dynamic hyperedge embeddings are added to the original embeddings to get $\mathbf{z}_{i^r}(t) = \mathbf{v}_{i^r}(t) + \mathbf{d}^{ch}_{i^r}$. Then we pass these through a self-attention layer (SAT) (refer 3.1.2) to get self

dynamic hyperedge embeddings,

$$\mathbf{d}_{i^r}^{sh} = \text{SAT}(\{\mathbf{z}_{1^r}(t), \ldots, \mathbf{z}_{k^r}(t)\}).$$

The complete dynamic hyperedge embeddings are obtained by combining them, $\mathbf{d}_{i^r}^{h} = \mathbf{d}_{i^r}^{sh} + \mathbf{d}_{i^r}^{ch}$.

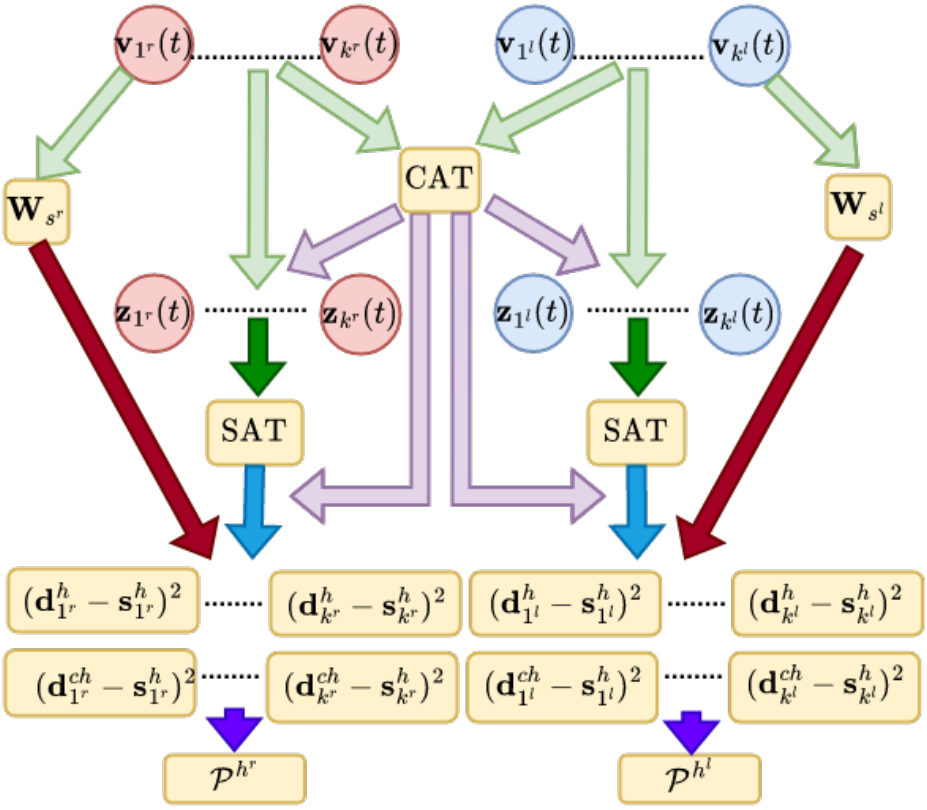


Figure 4.2: Hyperedge predictor architecture

We also create static hyperedge embeddings $\mathbf{s}_{i^r}^h = \mathbf{W}_{s^r}\mathbf{v}_{i^r}(t)$, where $\mathbf{W}_{s^r} \in \mathbb{R}^{d\times d}$ is a learnable parameter. Then we calculate the Hadamard power of the difference between static and dynamic hyperedge embedding pairs followed by a linear layer, and the final score $\mathcal{P}^{h^r}$ is calculated as shown below,

$$o_{i^r} = \mathbf{W}_{o^r}[(\mathbf{d}_{i^r}^h - \mathbf{s}_{i^r}^h)^2 || (\mathbf{d}_{i^r}^{ch} - \mathbf{s}_{i^r}^h)^2] + b_{o^r}, \text{Here, } || \text{ is the concatenation operator and}$$
$$\mathcal{P}^{h^r} = \frac{1}{k^r}\sum_{i^r=1}^{k^r} o_{i^r}. \tag{4.8}$$

Here, $\mathbf{W}_{o^r} \in \mathbb{R}^{1\times 2d}, b_{o^r} \in \mathbb{R}$ are the learnable parameters of the output layer. This equation

models the cross and self relations in the right hyperedge. Similarly, to incorporate the relations in left hyperedge, we find $\mathcal{P}^{h^l}$ with a different set of parameters, and we combine them to predict links as shown below,

$$f(\{\mathbf{v}_{1^r}(t), \mathbf{v}_{2^r}(t), \ldots, \mathbf{v}_{k^r}(t)\}\{\mathbf{v}_{1^l}(t), \mathbf{v}_{2^l}(t), \ldots, \mathbf{v}_{k^l}(t)\}) = \mathcal{P}^{h^r} + \mathcal{P}^{h^l}.$$

The model's entire block architecture and details of CAT and SAT are shown in Figure 4.2

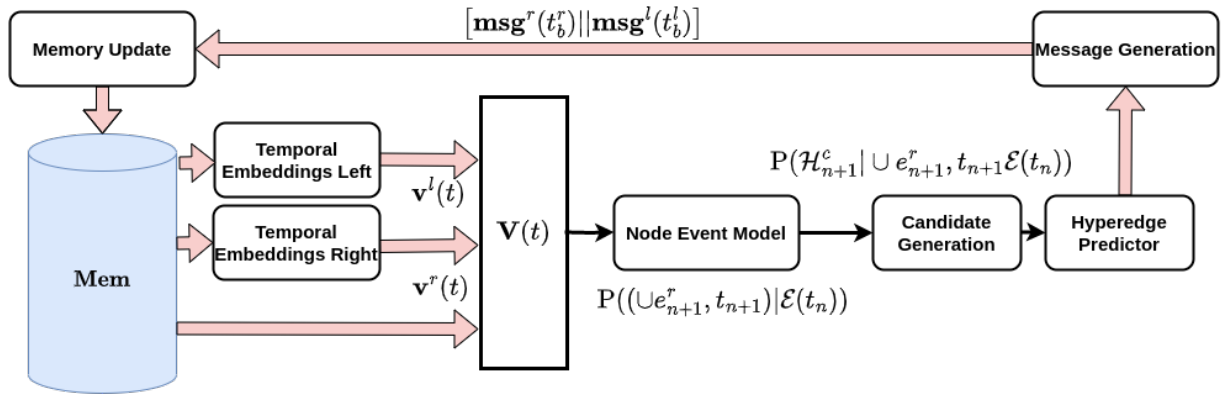


Figure 4.3: Neural Architecture of DHyperNodeTPP: We calculate the dynamic node embeddings $\mathbf{V}(t)$ by combining the embeddings from the Memory module with information from recent interactions where the node is involved in the left and right hyperedges. These dynamic embeddings are used to forecast nodes where events occur, followed by candidate hyperedge generation. Then the hyperlink prediction decoder in Section 4.2.2 is used to find the observed hyperedges.

4.2.3 Dynamic Node Embedding

The dynamic embedding of node v_i has the following architecture,

$$\mathbf{v}_i(t) = \tanh(\mathbf{W}_s\mathbf{Mem}_i + \mathbf{W}^r\mathbf{v}_i^r(t) + \mathbf{W}^l\mathbf{v}_i^l(t) + \mathbf{b}_v).$$

Here, $\mathbf{W}_s \in \mathbb{R}^{d\times d}, \mathbf{W}^r \in \mathbb{R}^{d\times d}, \mathbf{W}^l \in \mathbb{R}^{d\times d}$ and $\mathbf{b}_v \in \mathbb{R}^d$ are learnable parameters. The first term, $\mathbf{Mem}_i \in \mathbb{R}^d$ is the historical interaction information for node v_i stored in the Memory Module explained in Section 4.2.3.1. The second term, $\mathbf{v}_i^r(t)$ is the temporal embeddings calculated based on the recent interactions h where the node v_i is present in the right hyperedge, $v_i \in h^r$. Similarly, we calculate $\mathbf{v}_i^l(t)$ based on the interactions where a node is present in the left hyperedge. The architecture used for finding these embeddings is shown in Section 4.2.3.2.

4.2.3.1 Memory Module

Memory module $\mathbf{Mem} \in \mathbb{R}^{|\mathcal{V}| \times d}$ stores the historical interaction information for each node in the network till time t. This module is initialized with zero values, and when a node is involved in an interaction, its corresponding features are updated based on the output of the Message Generation Module. However, if we update the node embeddings with each interaction like in the previous work, in Chapter 3, we will not be able to scale the model to a large number of samples. So, we divide the interactions into batches of size $\mathcal{B}$ while maintaining their temporal order. Then we aggregate the temporal features of each node in the batch using the Message Generation stage. Then we update the memory embeddings of the node using a recurrent neural network as explained in Memory Update stage.

Message Generation. We calculate features for each node in all the interactions in a batch $(h_b, t_b)_{b=1}^{\mathcal{B}}$. For node v_i in the right hyperedge h_b^r of interaction (h_b, t_b), the features are the concatenation of its dynamic embedding $\mathbf{v}_i(t)$, dynamic hyperedge embeddings $\mathbf{d}_i^h$, and Fourier features (da Xu et al., 2020) calculated from the duration $t - t_i^p$ since the last memory update of that node happened at time t_i^p. This is stored as message vector as shown below,

$$\mathbf{msg}_i^r(t_b) = \left[\mathbf{v}_i(t)||\mathbf{d}_i^h||\boldsymbol{\psi}(t - t_i^p)\right].$$

Here, $\boldsymbol{\psi}(t - t_i^p) \in \mathbb{R}^d$ is the Fourier features based on functional encoding for the time $t - t_i^p$. This is achieved by learning a mapping function $\boldsymbol{\psi}(t) = [\cos(\omega_1 t + \phi_1), \ldots, \cos(\omega_d t + \phi_d)]$ with parameters $\{\omega_i\}_{i=1}^d$, and $\{\phi_i\}_{i=1}^d$ inferred from the data. Similarly, we calculate the message vectors $(\mathbf{msg}^l(.))$ for nodes in the left hyperedge.

Memory Update. The messages from the previous section are used to update the Memory Module entries of the corresponding nodes before the next batch. In our work, we use a GRU Cho et al. (2014) based recurrent neural networks for memory updating as shown below,

$$\mathbf{Mem}_i = \text{GRU}([\mathbf{msg}_i^r(t_b^r)||\mathbf{msg}_i^l(t_b^l)], \mathbf{Mem}_i), t_i^p = max(t_b^r, t_b^l). \tag{4.9}$$

Here, t_b^r, t_b^l are the latest event times for node v_i in previous batch. For a node if the messages are not available in the left or right, the corresponding features will have zero value.

4.2.3.2 Temporal Embeddings

To incorporate higher-order neighborhood information into the node embeddings, we use a temporal graph attention network to update the node representation with information from relevant historical interactions. Further, it also helps to avoid the staleness (Kazemi et al.,

2020) of embeddings in the Memory Module due to the absence of recent interactions involving the node by extracting information from other nodes that are previously involved with it. For a node v_i at time t, we find the recent $\mathcal{N}$ interactions involving the node in the right hyperedge, and we denote them as, $\mathcal{N}_{h^r}(t)$. Then for each hyperedge $(h, t) \in \mathcal{N}_{h^r}(t)$, we calculate the hyperedge features as shown below,

$$\mathbf{h}^r(t_i) = \frac{1}{|h^r|} \sum_{v_{i^r} \in h^r} \mathbf{W}_h^r \mathbf{Mem}_{i^r} + \frac{1}{|h^l|} \sum_{v_{i^l} \in h^l} \mathbf{W}_h^l \mathbf{Mem}_{i^l}. \tag{4.10}$$

Similarly, we find the recent $\mathcal{N}$ interactions where nodes are in the left hyperedge, $\mathcal{N}_{h^l}(t)$, and calculate hyperedge embeddings $\mathbf{h}^l(t_i)$. Then the right hyperedge embeddings are calculated using the temporal graph attention layer as shown below,

$$\begin{aligned} \mathbf{C}(t) &= \left[\mathbf{h}^r(t_1)||\boldsymbol{\psi}(t-t_1), \ldots, \mathbf{h}^r(t_{\mathcal{N}})||\boldsymbol{\psi}(t-t_{\mathcal{N}})\right], \\ \mathbf{q}(t) &= \mathbf{Mem}_i||\boldsymbol{\psi}(0), \\ \text{and } \mathbf{v}^r(t) &= \text{MultiHeadAttention}(\mathbf{q}(t), \mathbf{C}(t), \mathbf{C}(t)). \end{aligned} \tag{4.11}$$

The MultiHeadAttention (Vaswani et al., 2017) uses the node memory embeddings for query ($\mathbf{q}(t)$), and recent interaction embeddings as keys ($\mathbf{C}(t)$) and values ($\mathbf{C}(t)$). Similarly, we calculate $\mathbf{v}^l(t)$ using a separate MultiHeadAttention layer.

4.3 Experimental Settings and Results

4.3.1 Prediction Tasks

We create two tasks to evaluate our model's event forecasting capabilities and to compare its performance against baseline methods.

- **Interaction Type Prediction.** The goal of this task is to predict the type of event e_{n+1} occurring at time t_{n+1} given the past interaction $\mathcal{E}(t_n)$. It can be done using hyperedge predictor stage on candidate hyperedges, $\mathcal{H}^c$.

- **Interaction Time Prediction.** The goal of this task is to predict the next time of event t_{n+1} given the past interaction $\mathcal{E}(t_n)$ for the nodes of interest. It can be calculated using the time estimate $\Delta t_{n+1,i} = \exp(\mu_{n+1,i})$ for each node $v_i \in \cup e_{n+1}^r$.

Datasets	$\lvert\mathcal{V}\rvert$	$\lvert\mathcal{E}(T)\rvert$	$\lvert\mathcal{H}^r\rvert$	$\lvert\mathcal{H}^l\rvert$	T
Enron-Email	183	10,311	1,003	89	99,070
Eu-Email	800	208,403	11,897	744	69,459,254
Twitter	2,130	9,889	1,218	2,321	17,277
HepTh	451	9,882	8,384	1,352	21,532
ML-ArXiv	659	18,558	2,995	17,014	62,741
iAF1260b	1,668	2,084	2,010	1,985	N/A
iJO1366	1,805	2,253	2,174	2,146	N/A
USPTO	16,293	11,433	6,819	6,784	N/A

Table 4.2: Datasets used for Temporal and Static Directed Hypergraphs along with their vital statistics.

4.3.2 Datasets

Table 4.2 shows the statistics of both static and temporal-directed datasets. Here, $|\mathcal{V}|$ denotes the number of nodes, $|\mathcal{E}(T)|$ denotes the number of events, $|\mathcal{H}^r|$ denotes the number of unique right hyperedges, $|\mathcal{H}^l|$ denotes the number of unique left hyperedges, and T is the time span of the dataset. For the static datasets, there is no time span feature. A detailed description of each dataset is provided below.

Enron-Email. [1] This dataset comprises email exchanges between the employees of Enron Corporation. Email addresses are the nodes in the hypergraph. A temporal directed hyperedge is created by using the sender's address as the right hyperedge, the recipients' addresses as the left hyperedge, and the time of exchange as the temporal feature. Further, the hyperedge size is restricted to 25, the less frequent nodes are filtered out, and only exchanges involving Enron employees are included.

Eu-Email. (Yin et al., 2017) This is an email exchange dataset between members of a European research institution and the temporal directed hyperedge is defined exactly to that of the Enron-Email dataset. The hyperedge size is restricted to 25, and the less frequent nodes were filtered out.

Twitter. (Chodrow and Mellor, 2019) This dataset contains tweets about aviation industry exchanged between users for 24 hours. Here, directed hyperedge is formed by senders and receivers of a tweet.

[1] http://www.cs.cmu.edu/ enron/

HepTh. (Gehrke et al., 2003) This is a citation network from ArXiv, initially released as a part of KDD Cup 2003. It covers papers from January 1993 to April 2003 under the HepTh (High Energy Physics Theory) section. The author IDs are the nodes, the directed hyperedge is between the authors of the paper and the cited papers, and the publication time is taken as the time of occurrence. The hyperedge size is restricted to 25, and less frequent nodes are filtered out.

ML-ArXiv. This is a citation network created from papers under the Machine Learning categories - "cs.LG," "stat.ML," and "cs.AI" in ArXiv[1]. The definition of the hyperedge is identical to that of the HepTh dataset. All papers published before 2011 are filtered out, and nodes are restricted to authors who have published more than 20 papers during this period. Citations between these papers are considered to create a directed hyperedge.

iAF1260b. This is a metabolic reaction dataset from BiGG Models (King et al., 2015). It is a knowledge base of genome-scale metabolic network reconstructions.

iJO1366. Similar to the above dataset, this is a metabolic reaction dataset from BiGG Models.

USPTO. This dataset consists of reactions from USPTO-granted patents (Jin et al., 2017). This dataset has been processed and prepared to consist of organic reactions created using a subset of chemical substances containing only carbon, hydrogen, nitrogen, oxygen, phosphorous, and sulfur.

4.3.3 Baseline Models

HyperNodeTPP is an undirected version of our model DHyperNodeTPP with right and left hyperedge merged into a single hyperedge, $h = h^r \cup h^l$. This uses the same Dynamic embedding in Section 4.2.3 with only a single type of Temporal Embeddings 4.2.3.2 and the HyperSANN (Zhang et al., 2019) architecture for hyperedge prediction. **BHyperTPP** and **HyperTPP** are models taken from previous work in Chapter 3 for interaction forecasting. They use hyperedge events to model interactions and use a recurrent model to learn dynamic node embedding by updating them when an interaction occurs. Here, **HyperTPP** is developed for undirected hyperedge event forecasting, and **BHyperTPP** is developed for bipartite hyperedge events. **TGN** and **GAT** are pairwise models developed for forecasting pairwise interaction. Here, **TGN** is developed for forecasting events in a dynamic graph (Rossi et al., 2020).**GAT** is a static model that predicts the presence or absence of a link in a static graph (**?**).

[1]https://arxiv.org/help/bulk_data

4.3.4 Metrics of Evaluations

i) Mean Reciprocal Rank (MRR). For evaluating the performance of interaction type prediction at time t, we find the position of true hyperedges r_n against candidate negative hyperedges by ordering them in descending order of $\lambda_h(t)$. Here, negative hyperedges are calculated by replacing the entire left or right hyperedges with hyperedge of randomly sampled nodes and size. Then MRR is calculated as follows,

$$MRR = \frac{1}{N}\sum_{n=1}^{N}\frac{1}{r_n + 1}.$$

ii) Mean Absolute Error (MAE). For evaluating the performance of event time prediction, we find the average of absolute difference between true values t^{true} and estimated value $\hat{t}_{n,i}$ as follows,

$$MAE = \frac{1}{N}\sum_{n=1}^{N}\Big[\sum_{v_i \in \cup e_n^r}\frac{1}{|\cup e_n^r|}|t_n^{true} - \hat{t}_{n,i}|\Big].$$

iii) Recall. For evaluating the performance of predicting projected adjacency vectors, $\mathbf{a}_{n,i}^r, \mathbf{a}_{n,i}^l$, at time t, we calculate the proportion of true neighbors in the estimated adjacency vectors, $\hat{\mathbf{a}}_{n,i}^r, \hat{\mathbf{a}}_{n,i}^l$ as follows,

$$Recall = \frac{1}{2N}\sum_{n=1}^{N}\Big[\sum_{v_i \in \cup e_n^r}\frac{1}{|\cup e_n^r|}\frac{\left(\mathbf{a}_{n,i}^r\right)^{\mathrm{T}}\hat{\mathbf{a}}_{n,i}^r}{\left(\mathbf{a}^r{}_{n,i}\right)^{\mathrm{T}}\mathbf{a}_{n,i}^r}\Big] + \frac{1}{2N}\sum_{n=1}^{N}\Big[\sum_{v_i \in \cup e_n^r}\frac{1}{|\cup e_n^r|}\frac{\left(\mathbf{a}_{n,i}^l\right)^{\mathrm{T}}\hat{\mathbf{a}}_{n,i}^l}{\left(\mathbf{a}^l{}_{n,i}\right)^{\mathrm{T}}\mathbf{a}_{n,i}^l}\Big].$$

4.3.5 Settings

In all our experiments, we use the first 50% of hyperedges for training, 25% for validation, and the remaining 25% for testing. We use the Adam optimizer and the ExponentialLR scheduler with $\gamma = 0.95$ to train the models for 100 epochs with the batch size set to 128 and a learning rate of 0.001. Models have a fixed embedding size d of 64, and we use 20 negative hyperedges for each true hyperedge in the dataset. All the reported scores are the average of ten randomized runs, along with their standard deviation.

4.3.6 Results

Our proposed model DHyperNodeTPP is evaluated against previous works and an undirected baseline model, HyperNodeTPP. The results in Table 4.3 demonstrate that our models, HyperNodeTPP and DHyperNodeTPP, outperformed previous works, HyperTPP and BHyperTPP,

Methods		GAT	TGN	HyperTPP	BHyperTPP	HyperNodeTPP(ours)	DHyperNodeTPP(ours)
Enron-Email	MAE	*N/A*	*N/A*	35.77 ± 1.61	13.92 ± 0.35	**4.15 ± 0.01**	4.18 ± 0.02
	MRR	40.16 ± 7.15	42.22 ± 0.87	35.00 ± 3.94	36.09 ± 1.96	61.85 ± 0.01	**61.94 ± 0.01**
Eu-Email	MAE	*N/A*	*N/A*	20.58 ± 3.34	18.57 ± 1.62	12.23 ± 0.03	**12.22 ± 0.02**
	MRR	66.81 ± 0.02	69.15 ± 0.01	62.42 ± 1.79	55.34 ± 1.21	**75.95 ± 0.01**	68.05 ± 0.03
Twitter	MAE	*N/A*	*N/A*	21.58 ± 3.79	8.16 ± 0.70	**1.18 ± 0.01**	1.20 ± 0.01
	MRR	44.88 ± 0.05	55.20 ± 0.03	69.87 ± 0.72	70.19 ± 0.95	**84.47 ± 0.01**	82.12 ± 0.00
HepTh	MAE	*N/A*	*N/A*	16.19 ± 3.01	8.86 ± 0.08	1.25 ± 0.03	**1.20 ± 0.01**
	MRR	33.79 ± 8.95	51.70 ± 1.37	57.98 ± 0.82	57.40 ± 3.00	45.18 ± 0.02	**79.01 ± 0.01**
ML-ArXiv	MAE	*N/A*	*N/A*	29.94 ± 3.77	17.29 ± 0.57	1.25 ± 0.00	**1.24 ± 0.00**
	MRR	22.49 ± 4.31	37.58 ± 1.12	26.07 ± 0.29	28.13 ± 0.78	29.49 ± 0.00	**52.05 ± 0.01**

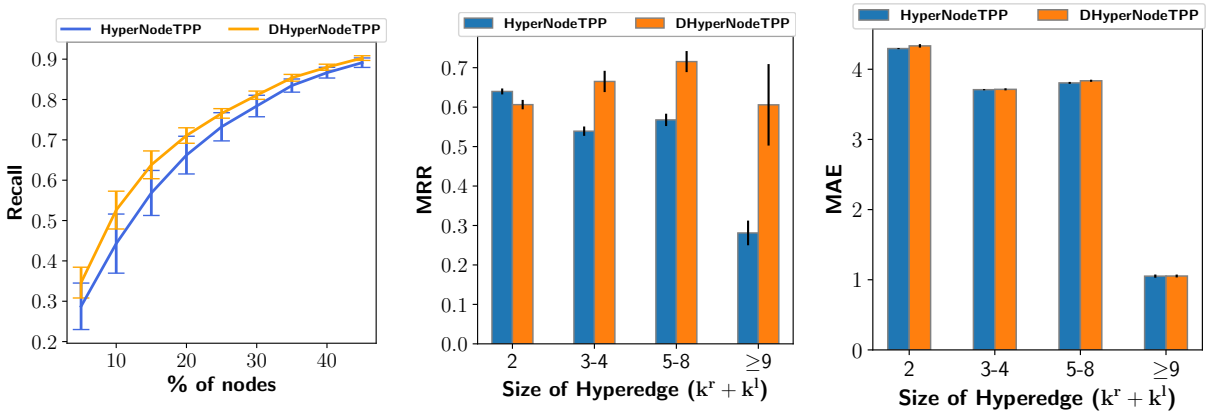
Table 4.3: Performance of dynamic directed hyperedge forecasting in interaction type and interaction time prediction tasks. The proposed model DHyperNodeTPP beats baseline models in almost all the tasks. Here, interaction type prediction is evaluated using MRR %; here higher value indicates better performance and interaction time prediction is evaluated using MAE, the lower value indicating better performance.

in interaction time prediction. This superior performance can be attributed to the fact that our models are trained to predict the next event on the nodes, as opposed to previous models that are trained to predict the next event on a hyperedge, which is more difficult and prone to error as they have to predict events of longer duration. However, in our case, events on nodes are more frequent and of shorter periods, hence the smaller error in the interaction time prediction.

Further, one can observe that the dynamic models perform much better than the static model GAT. This shows the need for dynamic models for real-world interaction forecasting. In our work, we achieve this using a Memory based dynamic node representation learning technique as explained in Section 4.2.3. We can also observe that models that use attention-based dynamic embeddings, HyperNodeTPP and DHyperNodeTPP, perform better than non-attentional models, BHyperTPP and HyperTPP, by comparing the performance in interaction type prediction tasks. This justifies using Temporal Embedding in Section 4.2.3.2 for dynamic node representation learning.

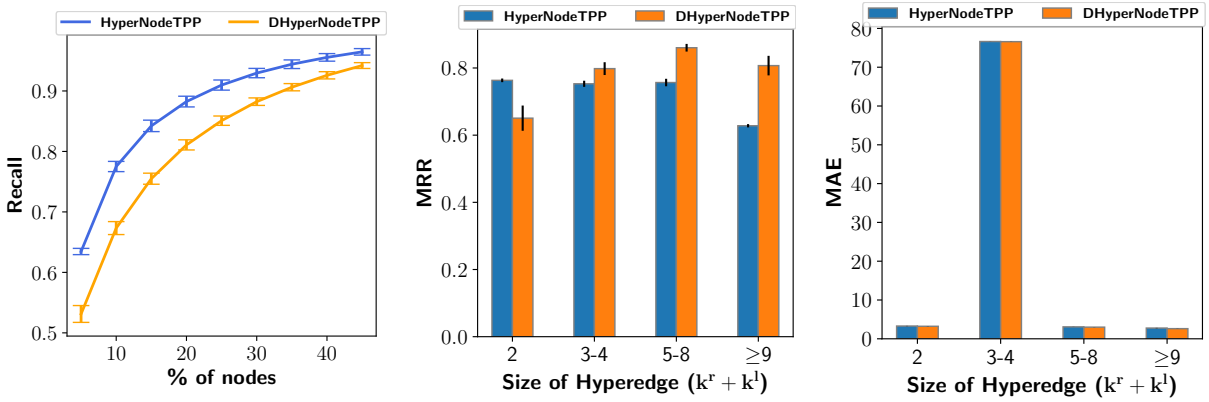
Comparing directed and undirected models. To observe the advantage of directed modeling, we compare the directed model DHyperNodeTPP with the undirected model HyperNodeTPP. Here, we see an average increase of 70% in the MRR metric for interaction type prediction in citation network based datasets Hepth and ML-ArXiv. The proposed model gives comparable performance to the undirected model for the other datasets. This is because in these datasets around 70% of interactions have only a single node in the left hyperedge, and HyperNodeTPP is performing well on these hyperedges of size two ($k^r + k^l = 2$).

To estimate the effect of hyperedge size on performance, we grouped hyperedges based on the total hyperedge size ($k = k^r + k^l$) and compared the performance of interaction type and time



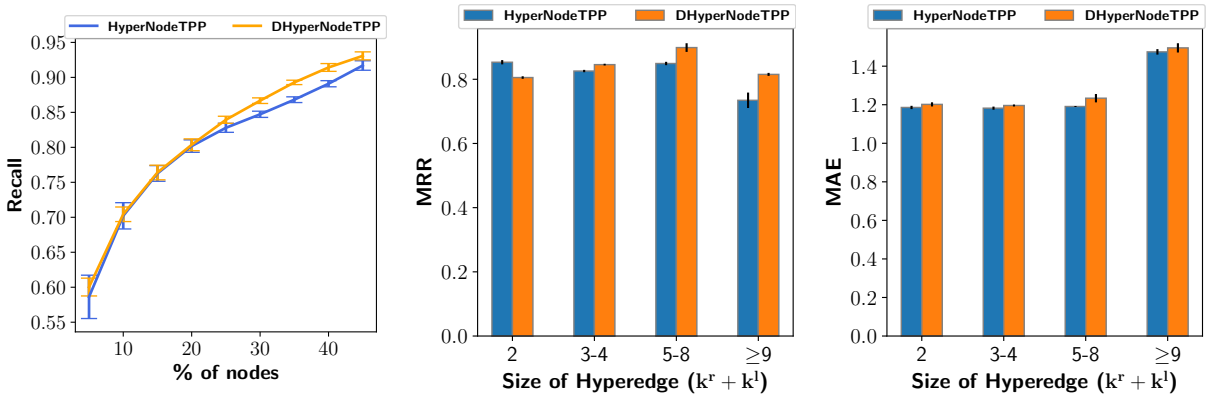
(a) Adjacency forecasting (b) Interaction type prediction (c) Interaction time prediction

Figure 4.4: Comparison of the performance of our directed and undirected model on different forecasting tasks on Enron dataset.



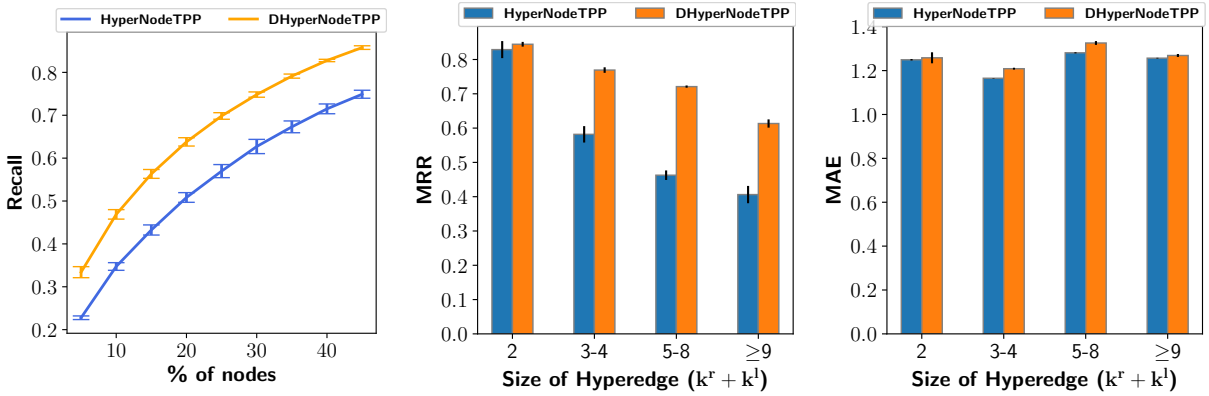
(a) Adjacency forecasting (b) Interaction type prediction (c) Interaction time prediction

Figure 4.5: Comparison of the performance of our directed and undirected model on different forecasting tasks on Eu-Email dataset.



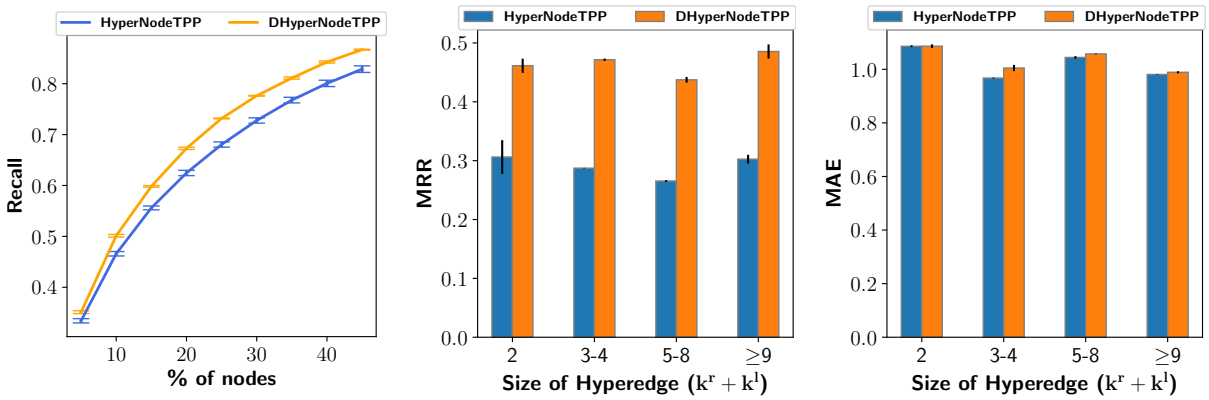
(a) Adjacency forecasting (b) Interaction type prediction (c) Interaction time prediction

Figure 4.6: Comparison of the performance of our directed and undirected model on different forecasting tasks on Twitter dataset.



(a) Adjacency forecasting (b) Interaction type prediction (c) Interaction time prediction

Figure 4.7: Comparison of the performance of our directed and undirected model on different forecasting tasks on HepTh dataset.

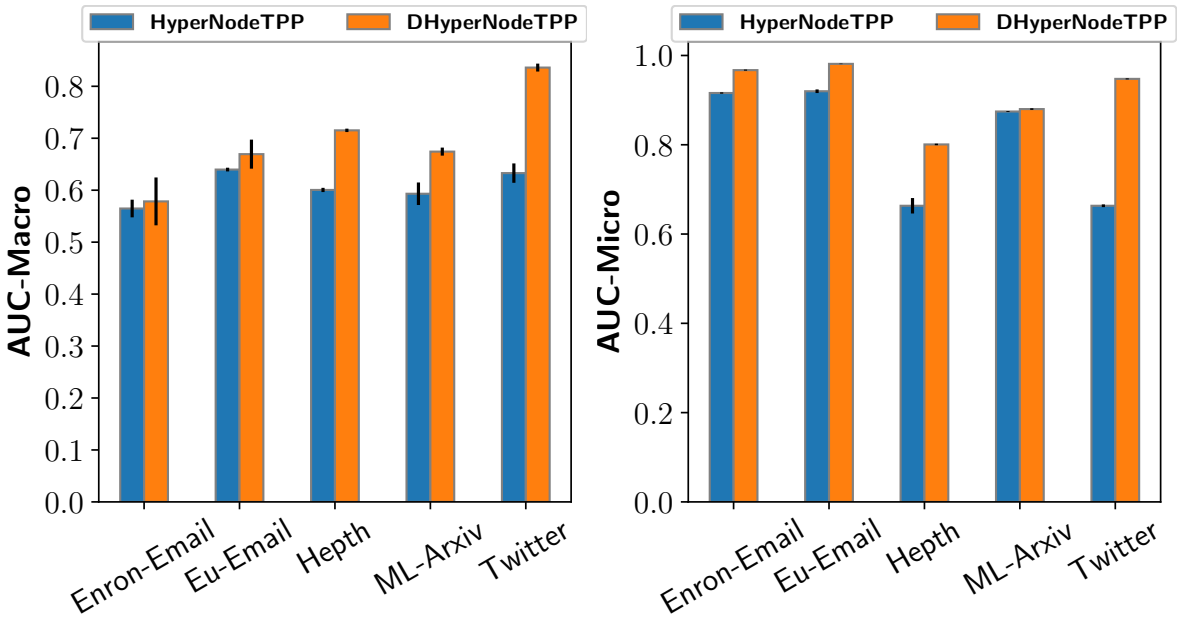


(a) Adjacency forecasting (b) Interaction type prediction (c) Interaction time prediction

Figure 4.8: Comparison of the performance of our directed and undirected model on different forecasting tasks on ML-ArXiv dataset.

prediction tasks for DHyperNodeTPP and HyperNodeTPP. Here, hyperedge groups have the following sizes $k = 2$, $3 \leq k \leq 4$, $5 \leq k \leq 8$, and ≤ 9. Figures 4.4b, 4.5b, 4.6b, 4.7b, and 4.8b show the comparison of MRR metrics for interaction type prediction tasks for DHyperNodeTPP and HyperNodeTPP models. For Enron-email, Eu-Email, and Twitter datasets, we can observe that HyperNodeTPP performs better than DHyperNodeTPP for hyperedge of size two. For hyperedges of size greater than two, DHyperNodeTPP outperforms HyperNodeTPP. In these datasets, Enron-Email, Eu-Email, and Twitter have around 80%, 81%, and 68% hyperedges of size two ($k = 2$), respectively, and HyperNodeTPP over-fits on these hyperedges which results in HyperNodeTPP outperforming DHyperNodeTPP when overall MRR is calculated, as shown in Table 4.3. For HepTh and ML-Arxiv datasets, DHyperNodeTPP outperforms HyperNodeTPP in all the hyperedge groups. Both models perform equally in all the hyperedge groups for the interaction time prediction, as shown in Figures 4.4c, 4.5c, 4.6c, 4.7c, and 4.8c. It is because event times are estimated at the node level, and hyperedge size does not affect the performance.

We also compared the models in the task of predicting projected adjacency vectors in the candidate hyperedge generation module. This is done by finding estimates for projected adjacency vectors by thresholding the estimated probability vectors, $\hat{\mathbf{a}}^r_{n,i} = \mathbb{I}_{\sigma(\boldsymbol{\theta}^r_{n,i}) > thres}, \hat{\mathbf{a}}^l_{n,i} = \mathbb{I}_{\sigma(\boldsymbol{\theta}^l_{n,i}) > thres}$, and recall is calculated as in Section 4.3.4. Here, *thres* is selected so that a fixed percentage of nodes will be presented as neighbors. Figures 4.4a, 4.5a, 4.6a, 4.7a and 4.8a, shows recall calculated for all datasets with respect to the percentage of nodes varying from 5% to 50% in the step of 5%. Here, we can see DHyperNodeTPP has more recall than HyperNodeTPP, especially when the percentage of allowed neighbors is small in datasets Enron-Email, HepTH, and ML-Arxiv.



(a) Hyperedge size prediction

(b) Hyperedge size prediction

Methods	HyperSAGNN	Bipartite Link Predictor	Ours
iAF1260b	60.1 ± 2.4	35.0 ± 1.4	**61.3 ± 0.5**
iJO1366	57.9 ± 1.8	36.2 ± 0.9	**59.0 ± 1.8**
USPTO	35.9 ± 1.8	37.8 ± 0.5	**38.1 ± 0.6**

Table 4.4: Performance of our hyperedge predictor in Section 4.2.2 to previous works on static directed hyperedge prediction. Here, the MRR metric is used to evaluate the performance.

Further, in the case of hyperedge size prediction, DHyperNode outperforms HyperNodeTPP in all the datasets. Figures 4.9a and 4.9b visualized the AUC-macro and AUC-micro (Fawcett, 2006) scores in a bar plot. Here, we can see our model DHyperNodeTPP outperforms HyperNodeTPP considerably. There is an average improvement of 14% and 9% in AUC macro and micro metrics, respectively.

Comparing hyperedge models with the pairwise models. The advantage of hyperedge models over pairwise edge models can be inferred by comparing models DHyperNodeTPP to TGN, which is the pairwise equivalent of DHyperNodeTPP. Our model DHyperNodeTPP has an average improvement of 37% in MRR metric over TGN in interaction type prediction.

Comparing different hyperedge prediction architectures. In Table 4.4, we compared our directed hyperedge predictor to previous works, HyperSANN Zhang et al. (2019) for undirected hyperedge prediction, and a Bipartite Link Predictor (refer to Section 3.3). Our method performs considerably better than previous architectures. We get an average improvement of

3.7% over the undirected model and 46.6% over the bipartite model in the MRR metric. This is because our link predictor models self-connections in both right and left hyperedges and also cross edges between them. The poor performance of the bipartite link predictor is because it focuses on modeling the cross connections and fails to model self connections in both right and left hyperedge due to the bipartite assumption.

4.4 Summary

In this work, we provide a solution for the problem of forecasting directed interactions based on historical data by presenting the model DHyperNodeTPP. The most previous works either reduce interactions to pairwise edges (da Xu et al., 2020, Gracious et al., 2021, Kumar et al., 2019) or ignore the directionality information in the interactions, as in Chapter 3. Further, the previous model HyperTPP that can forecast hyperedge suffers from scalability as it considers a marked temporal point process with each hyperedge as a separate mark, and the number of unique hyperedges is exponential to the number of nodes. So, to address this, we propose a multi-task approach involving three stages. The first stage is used to forecast nodes where the events will occur that are then consumed by the second task to forecast candidate hyperedges occurring at the event time, which is then used by the directed hyperlink prediction stage to filter out the true hyperedges. Since the number of nodes that observe an event at a particular time is very small, we could considerably reduce the search space for the possible hyperedge. Further, we provide a dynamic representation learning framework that can do batch training by using a Memory Module. This reduces computational complexity while training on very large datasets. We also provide an architecture for directed hyperedge prediction by combining three levels of relationship in it. This work also involves creating five datasets to show the advantage of our model over existing models for undirected and bipartite hyperedge forecasting models. Some exciting directions for future works are using advanced negative sampling techniques to make the model learn efficiently and provide good candidate hyperedges in a scalable way (Hwang et al., 2022). Also, to incorporate multi-hop information into the dynamic representation learning.

Chapter 5

RRHyperTPP for Events in Multi-Relational Recursive Hypergraph

So far, this thesis has mainly focused on using hyperedges to represent interactions of groups of entities/nodes. With the availability of richer datasets and better computational facilities, we can go beyond hypergraphs by incorporating a wide variety of information associated with the interaction. These ideas are being explored in recent works of topology-aware deep learning where higher-order network structures such as simplicial complexes, cell complexes, and multi-scale topological information are used for improving graph neural networks (Hajij et al., 2023, Horn et al., 2022). It has also shown that n-ary facts or multi-relational hyperedges are the natural data representations for Knowledge graphs, and models based on those perform better than that use pairwise multi-relational models (Wen et al., 2016, Fatemi et al., 2020).

In this chapter, we address the problem of forecasting interactions where group structure is inherent. For example, a directed hyperedge, where hyperedge involves interaction between two groups of entities in the source and destination. Here, one can represent the source and destination as hyperedges, which act as nodes inside the hyperedge representing the interaction. One can generalize this recursive group structure property of real-world interactions to incorporate a variable number of groupings. Examples of these types of interaction can be seen in email exchanges, where there are sender, recipients, and carbon-copied recipients (cced) addresses, as shown in Figure 5.1. Each group except the sender can have a variable number of addresses, and cced group is not always present in an email exchange. Similarly, in citation networks, the authors of a scientific article are ordered according to their contributions to the work. Here, the author's position is the grouping, and multiple authors can be in the same position.

We use a multi-relational recursive hyperedges structure to incorporate these complex inter-

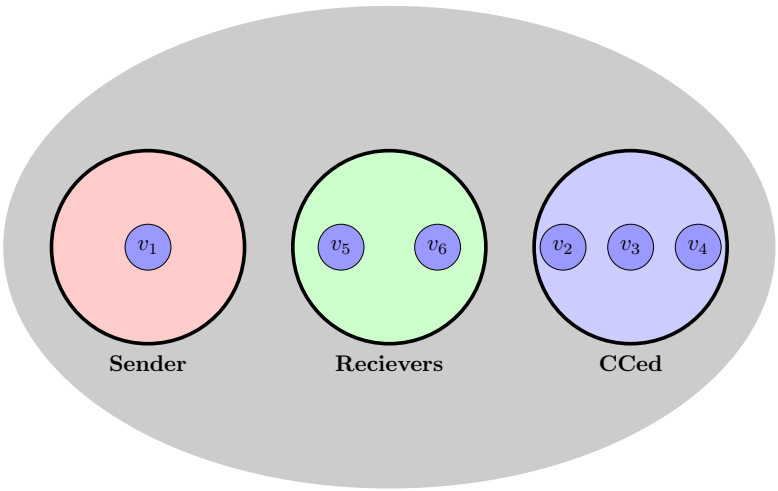


Figure 5.1: Email exchanges, where there are sender, recipients, and carbon-copied recipients (cced) addresses. Each group except the sender can have a variable number of addresses, and cced group is not always present in an email exchange.

actions. Here, hyperedges can act as nodes in other hyperedges (like cced or receiver groups in email interactions) and contain relation types indicating their role in the interactions. Then a TPP is defined with these hyperedges as event types with conditional intensity function parameterized with link predictor that uses dynamic node representations/embeddings and relation embeddings in the interactions. Furthermore, the previous approaches in TPP based interaction forecasting use negative sampling to approximate the loss function, which is intractable to calculate due to the huge number of possible event types in the TPP. However, these can introduce biases in training, especially in these complex interactions. To address these challenges, we propose the model *Relational Recursive Hyperedge Temporal Point Process* (RRHyperTPP) and the following are the contributions of our work,

- A contrastive learning strategy for higher-order interaction for hyperedge events in networks. This provides a more principled way of training our model without maximum likelihood estimation, thereby avoiding the computationally expensive survival function calculation. Our model performs better than the previous models that use negative sampling.

- New deep learning architecture for learning node representation from higher-order interaction. This involves two stages: interaction update and temporal drift. Interaction update is used to revise the node representation when an event involving it occurs by using the features of the type of interaction. Temporal drift is used to model the evolution of node representation during the interevent period using time projection based on JODIE (Ku-

mar et al., 2019) or Neural ODE (Chen et al., 2018) or time embeddings based on Fourier time features (da Xu et al., 2020).

- A new method for hyperedge link prediction for multi-relational recursive hypergraphs.
- Curated eight datasets for temporal multi-relational recursive hypergraphs.
- Extensive experiments are done to show the advantage of our model over existing state-of-the-art models.

5.1 Preliminaries and Problem Statement

A Hypergraph is denoted by the tuple $\mathcal{G} = (\mathcal{V}, \mathcal{H})$, where $\mathcal{V} = \{v_1, v_2, \ldots, v_{|\mathcal{V}|}\}$ is the set of nodes, $\mathcal{H}$ is the set of hyperedges. Here, $\mathcal{H} \subset \mathsf{C}(\mathcal{V})$, where $\mathsf{C}(\mathcal{V})$ is the powerset of $\mathcal{V}$. A Recursive Hypergraph $\mathcal{G} = (\mathcal{V}, \mathcal{H})$ is a generalization of Hypergraph, where hyperedges can contain nodes and other hyperedges. Let, $\mathcal{H}^l = \mathsf{C}(\cup_{i=0}^{l-1}\mathcal{H}^i)$ if $l \geq 1$, where $\mathcal{H}^0 = \mathsf{C}(\mathcal{V})$. A recursive hypergraph is of depth l if $\mathcal{H} \subset \mathcal{H}^l$. This work will focus on recursive hypergraphs of depth one and two. The following describes the temporal higher-order network used in this work.

Temporal Multi-relational Recursive Hypergraphs. A Temporal Multi-relational Recursive hypergraph is denoted by $\mathcal{G}(t) = (\mathcal{V}, \mathcal{H}, \mathcal{R}, \mathcal{E}(t))$, where $\mathcal{R}$ is the set of relations, and $\mathcal{E}(t) = \{(e_1, t_1), \ldots, (e_N, t_N)\}$ is the ordered set of historical events till time t with $e_i \in \mathcal{H}$ and N is number events occurred. Here, each hyperedge $h \in \mathcal{H}$ is formed of a set of recursive hyperedges, $h = h^l = \{(h_1^{l-1}, r_1^{l-1}), \ldots, (h_i^{l-1}, r_i^{l-1}), \ldots, (h_{k^{l-1}}^{l-1}, r_{k^{l-1}}^{l-1})\}$ of depth l. Here, $r_i^{l-1} \in \mathcal{R}$ is the relation of hyperedge h_i^{l-1} with respect to hyperedge h^l, and $h_i^{l-1} \in \mathcal{H}^{l-1}$. Here, k^{l-1} is the number of relational groups in hyperedge h, and $\mathcal{E}(t_a, t_b)$ is the ordered set of events observed during the interval $[t_a, t_b]$.

Using the above definition, one can represent a *Directed temporal hypergraph* by using relations indicating the source and destination groups, here $|\mathcal{R}| = 2$. Furthermore, including the node type attribute, one can extend this definition to a *Bipartite Temporal Hypergraph* where the nodes set is a union of two types, $\mathcal{V} = \mathcal{U}^r \cup \mathcal{U}^l$. Here, $\mathcal{U}^r$ and $\mathcal{U}^l$ are the two sets of nodes, and all interactions are between them. To use this framework for *Temporal Knowledge Graph* (TKGs) where interactions are represented as a triplet $h = (v^s, r, v^o))$ of the source entity v^s, target entity v^o, and relation r between them, here, $v^s, v^o \in \mathcal{V}$. This can be framed as a Multi-relational Recursive Hypergedge with $h = \{(h_0^0, r_0^0), (h_1^0, r_1^0)\}$. Here, $r_0^0 = r$, $r_1^0 = r^{-1}$ is the inverse relation, $h_0^0 = \{v^s\}$, and $h_1^0 = \{v^o\}$. Table 5.1 provides the summary of the notations used in this chapter.

Notation	Definition
$\mathsf{C}(.)$	Power set
$\mathcal{H}^l$	Hyperedges at depth l
l	Depth
$\mathcal{G}(t) = (\mathcal{V}, \mathcal{H}, \mathcal{R}, \mathcal{E}(t))$	A temporal multi-relation recursive hypergraph
$\mathcal{R}$	Set of relations
k^l	Number of depth l hyperedges in h
$h = \{(h_1^{l-1}, r_1^{l-1}), \ldots, (h_{k^{l-1}}^{l-1}, r_{k^{l-1}}^{l-1})\}$	Recursive hyperedge of depth l
r_i^{l-1}	Relation of hyperedge h_i^{l-1} with respect to hyperedge h^l
r^{-1}	Inverse relation
$\mathcal{U}^r, \mathcal{U}^l$	Right nodes and left nodes
v^s, v^o	Subject/Object entity
$\mathcal{L}_{NCE}$	Noise contrastive estimation
$\mathrm{Q}(.)$	Noise distribution
$\mathcal{E}^{n_q}$	Noise stream n_q
$n_q, n_q^{'}$	Noise index
N^q	Number of noise streams
$\varnothing$	Null event
$\underline{\lambda_h}(t)$	Combined intensity function
$\mathcal{H}^c$	Candidate hyperedges
$\mathcal{K}$	Kernel for KDE
w	Bandwith
λ^q	Rate of events
$\mathbf{i}_v^h$	Interaction features
$\mathbf{W}_t$	Temporal drift parameter
$f_{\nabla \mathbf{v}}$	Gradient of node embedding
$\bar{h}^0$	Expanded hyperedge at depth 0
$\bar{k}^0$	Size of expanded hyperedge at depth 0
$\mathbf{d}_{h_i^{l-1}}^{h^l}$	Dynamic node embedding at depth l for for hyperedge h_i^{l-1}
$\mathbf{h}_i^l$	Hyperedge embedding
$o_i^{\bar{h}}$	Output score for hyperedge link predictor

Table 5.1: Notations and their definitions

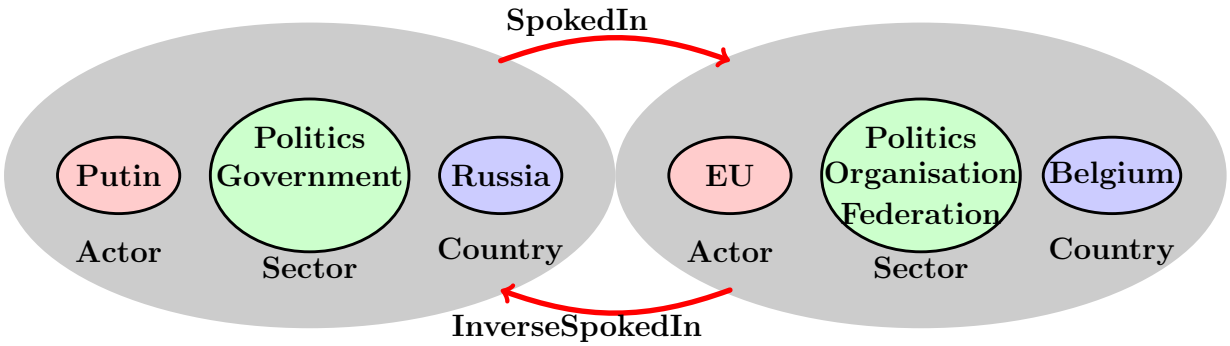


Figure 5.2: Real world interactions represented as Multi-Relational Recursive Hyperedges. Here, there is source and target hyperedge with three relations inside them and a relation type across them indicating the nature of the interaction. Here, relation **SpokedIn** indicates the action of speaking done by the source at the target, and **InverseSpokedIn** is the relation that indicates the target who spoke at the source.

Problem. Given the historical events $\mathcal{E}(t) = \{(e_1, t_1), \ldots, (e_n, t_n)\}$ until time t, we aim to forecast the next interaction (t_{n+1}, e_{n+1}) where $t_{n+1} > t_n$. Here, we want to estimate the time t_{n+1} and the type of interaction e_{n+1}.

This work mainly focuses on interactions formed from communications networks and real-world events stored in the form of source and target entities. The first type is represented using a recursive hyperedge graph of depth one $h = \{(h_i^0, r_i^0))\}_{i=0}^{k^0}$. Here, relations $\{r_i^0\}_{i=0}^3$ represent the sender, recipients, and carbon-copied recipients' addresses for the email exchange datasets, and $\{r_i^0\}_{i=0}^4$ represent the sender, receiver, retweeter, and retweeted nodes in the tweet based datasets. The second type of interaction is represented using depth two hyperedges $h = \{(h_0^1, r_0^1), (h_1^1, r_1^1)\}$ and $h_{0/1}^1 = \{(h_i^0, r_i^0)\}_{i=0}^3$. Here, h_0^1, h_1^1 are the source and target entities involved in the interaction, r_0^1 and r_1^1 are the inverse relation pair indicating the nature of the interaction, $\{r_i^0\}_{i=0}^3$ are relations indicating the actor, sector, country of the entity. Figure 5.2 shows an example of this type of interaction.

5.2 Relational Recursive HyperTPP Model

The probability of the occurrence of hyperedge event h at time t is defined as follows,

$$\mathrm{P}_h(t) = \lambda_h(t)\mathrm{S}_h(t_h^p, t). \tag{5.1}$$

Here, $\lambda_h(t)$ is the conditional intensity function of the temporal point process, $\mathrm{S}(t_h^p, t)$ is the survival function that denotes the probability that the event does not occur during the interval

$[t_h^p, t]$,

$$\mathrm{S}(t_h^p, t) = \exp\left(-\int_{t_h^p}^{t} \lambda_h(\tau)\mathrm{d}\tau\right), \tag{5.2}$$

Here, $t_h^p < t$ is the time of the last occurrence of hyperedge h, and it is initialized to zero for all the hyperedges at the beginning. The complete likelihood for observing $\mathcal{E}(T) = \{(e_1, t_2), \ldots, (e_N, t_N)\}$ in the interval $[0, T]$ can be written as,

$$\mathrm{P}(\mathcal{E}(T)) = \prod_{n=1}^{N} \mathrm{P}_{e_n}(t_n) \prod_{h \in \mathcal{H}} \mathrm{S}_h(t_h^\ell, T). \tag{5.3}$$

Here, t_h^ℓ is the last occurrence of hyperedge h with $t^\ell = 0$ for $h \in \mathcal{H}$ that is not observed in the training data.

$$\begin{aligned}
\mathrm{P}(\mathcal{E}(T)) &= \prod_{n=1}^{N} \lambda_{e_n}(t_n)\mathrm{S}(t_{e_n}^p, t_n) \prod_{h \in \mathcal{H}} \mathrm{S}_h(t_h^\ell, T), \\
\mathrm{P}(\mathcal{E}(T)) &= \prod_{n=1}^{N} \lambda_{e_n}(t_n) \prod_{h \in \mathcal{H}} \mathrm{S}_h(0, T).
\end{aligned} \tag{5.4}$$

For learning the parameters of conditional intensity $\lambda_h(t)$, loss is calculated by taking a negative log-likelihood as shown below,

$$\mathcal{NLL} = -\sum_{n=1}^{N} \log \lambda_{e_n}(t_n) + \sum_{h \in \mathcal{H}} \int_0^T \lambda_h(t)\mathrm{d}t. \tag{5.5}$$

However, direct minimization of the above loss function is computationally expensive due to the summation over all possible hyperedges in the survival function, which have an exponential number of possibilities, and also due to the integral over the entire duration of observation $\int_0^T(.)$. So, to avoid these difficulties in computing the likelihood, we use the noise contrastive estimation technique (Mei et al., 2020) of learning multivariate temporal point process as explained in Section 5.2.1.

Furthermore, defining separate conditional intensity functions for each hyperedge will lead to difficulty in learning, as the number of parameters will increase linearly to the number of hyperedges, which is exponential on the order of the number of nodes, as explained earlier. It leads to over-fitting while training and poor generalization in the test set. Hence, we define the conditional function as a positive function of dynamic embeddings nodes involved in it,

$\lambda_h(t) = f(h; \mathbf{V}(t), \mathbf{R})$, as explained in Section 5.4. Here, $\mathbf{V}(t)$ is the dynamic node embedding of nodes at time t, and $\mathbf{R}$ is the relation embeddings. In this model, the number of parameters will be linear to the number of nodes, as hyperedges that share nodes will share the respective parameters.

5.2.1 Learning and Inference

An alternate approach to MLE for learning parameters is to use noise contrastive estimation (NCE). To achieve this, we discretized the training period $[0, T]$ into intervals of size Δt and simulate N_q noise streams $\{\mathcal{E}^{n_q}(t, t+\Delta t)\}_{n_q=1}^{N_q}$ from the noise distribution $\mathrm{Q}(.|\mathcal{E}(t))$ in addition to observed data $\mathcal{E}(t, t+\Delta t)$. In the following section, we will also use the 0th index to indicate observed data $\mathcal{E}^0(T) = \mathcal{E}(T)$. Then the model parameters are trained to discriminate true, and noise events from the candidate set $\{\mathcal{E}^{n_q}(t, t+\Delta t)\}_{n_q=0}^{N_q}$. This can be written as follows,

$$\log \frac{\mathrm{P}\left(\mathcal{E}^0(t, t+\Delta t)|\mathcal{E}(t)\right) \prod_{n_q=1}^{N_q} \mathrm{Q}(\mathcal{E}^{n_q}(t, t+\Delta t)|\mathcal{E}(t))}{\sum_{n_q=0}^{N_q} \mathrm{P}(\mathcal{E}^{n_q}(t, t+\Delta t)|\mathcal{E}(t)) \prod_{n_q' \neq n_q} \mathrm{Q}(\mathcal{E}^{n_q'}(t, t+\Delta t)|\mathcal{E}(t))}. \tag{5.6}$$

However, we still need to calculate the integral $\sum_{h\in\mathcal{H}} \int_t^{t+\Delta t}$ in the probability density function, which is still computationally expensive. The solution is to shrink the interval to an infinitesimal width of $\mathrm{d}t$ when $\lim \mathrm{d}t \to 0$. Then one can rewrite the Equation 5.6 as,

$$\log \frac{\mathrm{P}\left(\mathcal{E}^0(t, t+\mathrm{d}t)|\mathcal{E}(t)\right) \prod_{n_q=1}^{N_q} \mathrm{Q}(\mathcal{E}^{n_q}(t, t+\mathrm{d}t)|\mathcal{E}(t))}{\sum_{n_q=0}^{N_q} \mathrm{P}(\mathcal{E}^{n_q}(t, t+\mathrm{d}t)|\mathcal{E}(t)) \prod_{n_q' \neq n_q} \mathrm{Q}(\mathcal{E}^{n_q'}(t, t+\mathrm{d}t)|\mathcal{E}(t))}. \tag{5.7}$$

Here, in the interval $[t, t+\mathrm{d}t]$, an event will be observed $\mathcal{E}^0(t, t+\mathrm{d}t) = \{(h, t)\}$ or a null event will be observed $\mathcal{E}^0(t, t+\mathrm{d}t) = \{(\varnothing, t)\}$ with respect to the distribution $\mathrm{P}\left(\mathcal{E}^0(t, t+\mathrm{d}t)|\mathcal{E}(t)\right)$. So, the probability distribution should satisfy the following inequality at the interval $[t, t+\mathrm{d}t]$,

$$\begin{aligned} \sum_{h\in\mathcal{H}} \mathrm{P}\left(\mathcal{E}^0(t, t+\mathrm{d}t) = (h_i, t)|\mathcal{E}(t)\right) + \mathrm{P}\left(\mathcal{E}^0(t, t+\mathrm{d}t) = (\varnothing, t)|\mathcal{E}(t)\right) &= 1, \\ \sum_{h\in\mathcal{H}} \lambda_{h_i}(t)\mathrm{d}t + \mathrm{P}\left(\mathcal{E}^0(t, t+\mathrm{d}t) = (\varnothing, t)|\mathcal{E}(t)\right) &= 1. \end{aligned} \tag{5.8}$$

Here, $\mathrm{P}\left(\mathcal{E}^0(t, t+\mathrm{d}t) = \{(h, t)\}|\mathcal{E}(t)\right) = \lambda_h(t)\mathrm{d}t$ from the temporal point process conditional intensity function definition, and $\mathrm{P}\left(\mathcal{E}^0(t, t+\mathrm{d}t) = (\varnothing, 1)|\mathcal{E}(t)\right) = 1$ when $\mathrm{d}t \to 0$. Similarly, we can define $\mathrm{Q}\left(\mathcal{E}^{n_q}(t, t+\mathrm{d}t)|\mathcal{E}(t)\right)$ using its definition of conditional intensity function. As a result of this, our objective function in Equation 5.7 where there are no events in the observed stream $\mathcal{E}^0(t, t+\mathrm{d}t) = (\varnothing, t)$ and in the N_q noise streams $\{\mathcal{E}^{n_q}(t, t+\mathrm{d}t) = \{(\varnothing, t)\}\}_{n_q=1}^{N_q}$ becomes

$\log \frac{1}{N_q+1}$ independent of model parameters and can be removed from loss. Then when event h occurs in the true data stream or the noise data stream, the loss can be written as follows,

$$\log \frac{\lambda_h(t)}{\lambda_h(t) + \lambda_h^q(t) N_q} \text{ if } \mathcal{E}(t, t+\mathrm{d}t) = \{(h,t)\} \text{ else,}$$
$$\log \frac{\lambda_h^q(t)}{\lambda_h(t) + \lambda_h^q(t) N_q} \text{ if } \exists \mathcal{E}^{n_q}(t, t+\mathrm{d}t) = \{(h,t)\} \text{ for } n_q = 1 \text{ to } N_q \ . \tag{5.9}$$

Then the complete loss function for the noise contrastive estimation can be written as follows,

$$\mathcal{L}_{NCE} = -\mathbf{E}_{\mathcal{E}(T)\sim \mathrm{P}(.), \{\mathcal{E}^{n_q}(T)\}_{n_q=1}^{N_q} \sim \mathrm{Q}(.)} \Big[\sum_{\mathcal{E}(t,t+\mathrm{d}t)=\{(h,t)\}} \log \frac{\lambda_h(t)}{\underline{\lambda_h}(t)} + \sum_{n_q=1}^{N_q} \sum_{\mathcal{E}^{n_q}(t,t+\mathrm{d}t)=\{(h,t)\}} \log \frac{\lambda_h^q(t)}{\underline{\lambda_h}(t)} \Big]. \tag{5.10}$$

Here, $\underline{\lambda_h}(t) = \lambda_h(t) + \lambda_h^q(t) N_q$. In the above equation, we contrast the samples from the true distribution, which is the observed data, to the N_q independently sampled noise stream conditioned on the observed historical data. A more detailed explanation of this contrastive loss function and the optimality of this technique can be found in work by Mei et al. (2020).

Additionally, to direct the gradients toward better model parameters, we add a classification-based noise contrastive loss at the time of the event as follows,

$$\mathcal{L}_{NCE}^{s} = \sum_{(e_n, t_n) \in \mathcal{E}(T)} \log \frac{\lambda_{e_n}(t)}{\lambda_{e_n}(t) + \sum_{h \in \mathcal{H}_{e_n}^{neg}} \lambda_h(t)}. \tag{5.11}$$

Here, $\mathcal{H}_{e_n}^{neg}$ are the negative samples generated by negative sampling for the hyperedge e_n.

5.2.2 Noise Distribution

Here, we propose a method for simulating noise sequences by modeling the conditional distribution $\mathrm{Q}(\mathcal{E}^{n_q}(t, t+\mathrm{d}t) = \{(h,t)\} | \mathcal{E}(t_n))$. The thinning algorithm is a popular method for simulating a non-homogeneous Poisson process (Chen, 2016). This way of sampling is computationally expensive if the acceptance probability is low. Moreover, modeling each hyperedge as a non-homogeneous Poisson process does not avoid the computational complexity of having many event types.

To overcome these limitations and accelerate training, we model the rate of events λ^q using a stochastic process that is independent of the event type and time. The probability density function of this process is learned from the observed event times in the training data $\mathcal{E}(T)$ using kernel density estimation, $\lambda^q \sim \frac{1}{Nw} \sum_{i=1}^{N} \mathcal{K}(\lambda^q - \frac{1}{t_i - t_{i-1}}; w)$ (Silverman, 1986). Here, $\mathcal{K}$ is

the kernel, and w is the bandwidth of the kernel. We use the Gaussian kernel, and bandwidth is learned from the dataset. This will allow for a closed-form sampling of noise event times. Then we model the type of event at each noise sample position as uniformly sampled from the candidate hyperedges $\mathcal{H}^c$. Then $\lambda_h^q = \frac{1}{|\mathcal{H}^c|}\lambda^q$ and conditional distribution of noise is as follows,

$$\mathrm{Q}(\mathcal{E}^{n_q}(t, t+\mathrm{d}t) = \{(h,t)\}|\mathcal{E}(t_n)) = \frac{1}{|\mathcal{H}^c|}\lambda^q \exp\left(-\lambda^q(t-t_n)\right).$$

Algorithm 5 Training procedure of RelHyperTPP

Input: $\mathcal{G}(T)$.
while not convergence **do**
  $\mathcal{H}^{neg} \sim$ CorruptionModel($\mathcal{H}$).
  $\mathcal{H}^c = \mathcal{H} \cup \mathcal{H}^{neg}$.
  Set $t = t_0 = 0$, $n = 1$, $\mathcal{L}_{NCE} = 0$.
  for $n \leq N$ **do**
    while $t < t_n$ **do**
      Sample $\lambda^q(t) \sim \frac{1}{Nw}\sum_{i=1}^{N}\mathcal{K}(\lambda^q - \frac{1}{t_i - t_{i-1}}; w)$.
      Sample $\Delta t \sim$ exponenetial($N_q\lambda^q(t)$).
      Sample $h \sim$ Uniform($\mathcal{H}^c$).
      Next event time $t = t + \Delta t$.
      Loss = Loss $- \log \frac{\lambda_h^q(t)}{\underline{\lambda_h}(t)}$
    end while
    $\mathcal{H}_{e_n}^{neg} \sim$ CorruptionModel(e_n)
    Loss = Loss $- \log \frac{\lambda_{e_n}(t)}{\underline{\lambda_{e_n}}(t)}$
    Loss = Loss $- \log \frac{\lambda_{e_n}(t)}{\lambda_{e_n}(t) + \sum_{h \in \mathcal{H}_{e_n}^{neg}} \lambda_h(t)}$
    $n = n + 1$.
    If $n \mod \mathcal{B} = 0$, Update the model parameters by AdamW Optimizer and set Loss = 0.
  end for
end while

The set of candidate hyperedges, denoted as $\mathcal{H}^c$, is formed by combining the true hyperedges observed in the training data with the negative hyperedges generated through the corruption of observed hyperedges. For a hyperedge of depth one, we first expand the depth zero hyperedges as a tuple of node and relation pairs in ascending order along with an end state indicator. Then

we learn the categorical distribution of each position condition on all the previous relations, along with the end state. Then for generating a negative hyperedge, we first sequential sample the relations till the end state is observed and randomly populate each position in the negative hyperedge with nodes. This is done by filling each position with nodes from the true hyperedge or randomly sampled nodes without nodes being repeated if they have the same relation value. This is done so that, at most, half of the negative hyperedge nodes are from true hyperedge, and the rest are filled in randomly, thereby ensuring diversity in negative samples. For depth two hyperedge datasets, we randomly corrupt any of the depths one hyperedge within them by the above procedure to generate negative hyperedges. Here, for each observed hyperedge $h \in \mathcal{H}$, we generate $N^e = |\mathcal{H}_h^{neg}|$ negative hyperedges.

For generating N_q noise streams, we multiply the noise distribution intensity function by N^q, $\lambda^q(t)N_q$, and simulate samples. This scaling will not affect NCE loss in Equation 5.10, as all the noise streams have the same intensity values.

Algorithm 5 summarizes the entire training procedure of the model. The CorruptionModel(.) is the negative hyperedge generation function, and Uniform($\mathcal{H}^c$) uniformly samples a hyperedge from the candidate hyperedges, $\mathcal{H}^c$. In the following section, we explain the dynamic node representation learning used to parameterize the conditional intensity function of the model. In our implementation, we use the same negative samples to generate the noise stream, and in the supervised noise contrastive loss in Equation 5.11, $\mathcal{H}^n = \cup_{h\in\mathcal{H}}\mathcal{H}_h^{neg}$.

5.3 Dynamic Node Representation

The dynamic node representation of the nodes $\mathbf{V}(t) \in \mathbb{R}^{|\mathcal{V}|\times d}$ in the networks are learned using a continuous time recurrent-neural network model using two stages, i) Node Update and ii) Drift, for node evolution.

5.3.1 Node Update

This stage is used to update the node embeddings of a node v when it is involved in an interaction $h = \{(h_1^{l-1}, r_1^{l-1}), \ldots, (h_i^{l-1}, r_i^{l-1}), \ldots, (h_{k^1}^{l-1}, r_{k^1}^{l-1})\}$ at time t. The update equation uses a recurrent neural network based architecture that updates the node's previously stored embedding based on the features of interaction calculated as follows,

$$\begin{aligned} \mathbf{i}_{v_i}^h &= \text{MLP}(\left[\mathbf{d}_{v_i}^h; \mathbf{v}_i(t^-); \mathbf{\Psi}(t - t_{v_i}^p)\right]) \\ \mathbf{v}_i(t^+) &= \text{RNN}(\mathbf{v}(t_{v_i}^p), \mathbf{i}_{v_i}^h). \end{aligned} \tag{5.12}$$

Here, $v \in h^0, (h^0, r^0) \in \ldots \in h$, MLP is a two-layer multi-layer perceptron which has the following form,

$$\mathbf{i}_v^h = \mathbf{W}_1^i \tanh\left(\mathbf{W}_0^i \left[\mathbf{d}_v^h; \mathbf{v}(t^-); \mathbf{\Psi}(t - t_v^p)\right]\right), \tag{5.13}$$

with learning parameters, $\mathbf{W}_0^i \in \mathbb{R}^{d*(l+3)\times d*(l+3)/2}$, $\mathbf{W}_1^i \in \mathbb{R}^{d*(l+3)/2\times d}$, and $\mathbf{d}_v^h \in \mathbb{R}^d$ is the dynamic embedding calculated as a function of embeddings of other nodes involved in the interaction, $\mathbf{d}_v^h = f_{dyn}(h; \mathbf{V}(t), \mathbf{R})$. The architecture of $f_{dyn}(\cdot)$ is shared with hyperedge link prediction as explained in Section 5.4. The second term $\mathbf{\Psi}(t - t_v^p) \in \mathbb{R}^d$ is the Fourier features for a time calculated based on the duration since an event was observed on v, $(t - t_v^p)$. Here, t_v^p is the recent time node v is involved in an interaction, and Fourier time features da Xu et al. (2020) are defined as $\mathbf{\Psi}(t) = [\cos(\omega_1 t + \phi_1), \ldots, \cos(\omega_d t + \phi_d)]$,where $\{\omega_i\}_{i=1}^d$, and $\{\phi_i\}_{i=1}^d$ are learnable parameters. When v is involved in multiple positions in hyperedge h, an average of $\mathbf{i}_v^h$ is taken to update the node embedding.

5.3.2 Temporal Drift.

This stage is used for modeling the evolution of the nodes during the inter-event period. This helps to avoid the staleness of node embeddings due to not observing any event for a long duration (Kazemi et al., 2020).

Time Projection. In this technique, we project the node embeddings from the node update stage based on the elapsed time since the last update as follows,

$$\mathbf{v}(t) = (1 + \mathbf{W}_t(t - t_v^p)) \circ \mathbf{v}(t_v^p). \tag{5.14}$$

Here, $\mathbf{W}_t \in \mathbb{R}^{d\times 1}$ is a learnable parameter, and $\circ$ denotes the Hadamard product. This version of the embedding projection was introduced in the work JODIE (Kumar et al., 2019).

Time Embeddings. Here, we use Fourier time features to model the evolution of the nodes during the interevent time as follows,

$$\mathbf{v}(t) = \tanh(\mathbf{W}_s \mathbf{v}(t_v^p) + \mathbf{W}_t \mathbf{\Psi}(t - t_v^p)). \tag{5.15}$$

Here, $\mathbf{W}_s, \mathbf{W}_t \in \mathbb{R}^{d\times d}$ are learnable parameters. This form of drift stage is followed in previous work Chaper 3.

Neural ODE. The node embedding evolution is modeled using a neural ordinary differential equation as follows,

$$\mathbf{v}(t) = \mathbf{v}(t_v^p) + \int_{t_v^p}^{t} f_{\nabla \mathbf{v}} \left(\mathbf{v}(t), t - t_v^p \right) \mathrm{d}t. \tag{5.16}$$

Here, $f_{\nabla \mathbf{v}}$ is the gradient of the node embedding $\mathbf{v}(t)$. We implement it using a multi-layer neural network that takes current node embedding and time elapsed since the last update as inputs to the network, as shown below,

$$f_{\nabla \mathbf{v}} \left(\mathbf{v}(t), t - t_v^p \right) = \mathbf{W}_1^t \tanh \left(\mathbf{W}_0^t \left[\mathbf{v}(t); \mathbf{W}_t (t - t_v^p) \right] \right). \tag{5.17}$$

Here, $\mathbf{W}_t \in \mathbb{R}^{1 \times d}, \mathbf{W}_0^t \in \mathbb{R}^{2d \times d}, \mathbf{W}_1^t \in \mathbb{R}^{d \times d}$. Neural ordinary differential equations are infinite depth deep learning models that are suited for modeling continuous dynamical systems. We use the adjoint sensitivity method proposed in (Chen et al., 2018) for reducing memory cost during back-propagation.

5.4 Hyperedge Link Prediction

Given the node embeddings and relation embeddings of a hyperedge, we use the quey-key-value attention mechanism used in transformer models (Vaswani et al., 2017) for hyperedge link prediction. For this, we first expand hyperedges at depth 0 as a tuple of node and relation pair and create an expanded depth one hyperedge $\bar{h}^0 = \{(v, r^0)\}_{v \in h^0, (h^0, r^0) \in h^1}$. Here, the relation is repeated for each node in the hyperedges. Then we create embeddings for query, key, and value, $\{\mathbf{v}_i^{\mathrm{q}}(t) = \mathbf{v}_i^{\mathrm{v}}(t) = [\mathbf{v}_i(t); \mathbf{r}_i^0]\} \forall (v_i, r_i^0) \in \bar{h}^0$, then we calculate the dynamic hyperedge embedding (here, dynamic is used to indicate the hyperedge dependent embeddings $\mathbf{d}_*^*$'s) for each node as follows,

$$\mathbf{d}_v^{\bar{h}^0} = \mathrm{MultiHeadAttention}(\{\mathbf{v}^{\mathrm{q}}(t)\}, \{\mathbf{v}_j^{\mathrm{k}}(t)\}_{j=1}^{\bar{k}^0}, \{\mathbf{v}_j^{\mathrm{v}}(t)\}_{j=1}^{\bar{k}^0}). \tag{5.18}$$

Here, MultiHeadAttention$(\cdot)$ is a multi-head attention layer. Based on these, we create the depth one hyperedge embeddings by averaging out all the above node embeddings $\mathbf{h}_i^1 = \sum_{v \in \bar{h}_i^0} \frac{\mathbf{d}_v^{\bar{h}_i^0}}{|\bar{h}_i^0|}$. Then for the hyperedge at depths $2 \leq \ell \leq l$, we apply the following self-attention

layer with query, key, and value as $\{\mathbf{v}_i^{\mathrm{q}}(t) = \mathbf{v}_i^{\mathrm{k}}(t) = [\mathbf{h}_i^{\ell-1}(t); \mathbf{r}_i^{\ell-1}]\} \forall (h_i^{\ell-1}, r_i^{\ell-1}) \in h^{\ell}$,

$$\begin{aligned} \mathbf{d}_{h_i^{\ell-1}}^{h^{\ell}} &= \text{MultiHeadAttention}(\{\mathbf{v}_i^{\mathrm{q}}(t)\}, \{\mathbf{v}_j^{\mathrm{k}}(t)\}_{j=1}^{k^{\ell-1}}, \{\mathbf{v}_j^{\mathrm{v}}(t)\}_{j=1}^{k^{\ell-1}}), \\ \mathbf{h}^{\ell} &= \sum_{(h_i^{\ell-1}, r_i^{\ell-1}) \in h^{\ell}} \frac{\mathbf{d}_{h_i^{\ell-1}}^{h^{\ell}}}{|h^{\ell}|}. \end{aligned} \tag{5.19}$$

Then conditional density is parameterized by the difference between the hyperedge embedding and dynamic hyperedge embedding of the $l-1$th layer as follows,

$$\begin{aligned} o_i^h &= \mathbf{W}_o(\mathbf{h}_i^{l-1} - \mathbf{d}_{h_i^{l-1}}^{h^l})^2 + b_o \\ \lambda_h(t) &= \text{Softplus}\Big(\sum_{(h_i^{l-1}, r_i^{l-1}) \in h} o_i^h\Big). \end{aligned} \tag{5.20}$$

The dynamic hyperedge embedding used in the interaction update is defined as follows,

$$\mathbf{d}_v^h = [\mathbf{v}(t); \mathbf{r}^0; \mathbf{d}_v^{\bar{h}^0}; \mathbf{d}_{h^1}^{h^2}; \ldots; \mathbf{d}_{h^{l-1}}^{h^l}]$$

, here $v \in h^0, h^0 \in h^1, \ldots, h^{l-1} \in h$.

5.5 Experimental Settings and Results

5.5.1 Datasets

Table 5.2 shows the vital statistics of the datasets used in this work. Enron is an email exchange dataset, and Twitter, Boston, Obama, Pope, and Cannes are datasets based on Twitter exchanges. These datasets created from the works by Chodrow and Mellor (2019), Domenico and Altmann (2020), Omodei et al. (2015) and are all depth one hyperedge. ICEWS-India, and ICEWS-Nigeria are depth two hyperedges created from events stored in Integrated Crisis Early Warning System data for countries India and Nigeria (Boschee et al., 2015). For preprocessing, we have filtered out the nodes based on frequency, and all the timestamps start from zero and are scaled down so that the mean interevent time is around one. Here, $\Delta t(mean)$, $\Delta t(max)$, and $\Delta t(min)$ are the mean, maximum, and minimum interevent time.

5.5.2 Tasks and Evaluation Metrics

We use two tasks to evaluate our model's performance and compare it against baseline models: **(i)** Interaction Type Prediction and **(ii)** Interaction Time Prediction.

In the first task, we are trying to predict which type of hyperedge occurs at time t given

Datasets	$\lvert\mathcal{V}\rvert$	$\lvert\mathcal{E}(T)\rvert$	$\lvert\mathcal{R}\rvert$	T	$\Delta t(mean)$	$\Delta t(max)$	$\Delta t(min)$
Enron	98	10,355	3	10,443.0	1.02	893.45	0.006
Twitter	2,714	52,383	4	54,028	1.54	25.64	0.625
Boston	2,400	20,070	4	20,069	1.03	37.13	0.044
Obama	1,721	22,690	4	22,689	1.02	1628.61	0.021
Pope	6,648	67,779	4	67,778	1.07	68.97	0.047
Cannes	672	9,078	4	9,077	1.00	58.02	0.007
ICEWS-India	1,066	86,609	201	364	1.00	1.0	1.0
ICEWS-Nigeria	894	6,95,433	181	715	1.0	2.94	0.98

Table 5.2: Datasets used for Temporal Multi-Relational Recursive Hypergraphs and their vital statistics.

the history. This can be estimated by finding the hyperedge that has the maximum probability $\mathrm{P}(\mathcal{E}(t, t+\mathrm{d}t) = (h, t))|\mathcal{E}(t))$, which is equal to finding the hyperedge with maximum conditional intensity function as per definition of TPP, $\hat{h} = \arg\max_h \lambda_h(t)$. For evaluation, we calculate the Area Under the Curve (AUC) metric (Fawcett, 2006) by using the CorruptionModel to create 20 false positive samples for each true positive sample and then calculate the area under the curve. Here, better performance is indicated by values close to one.

For the second task, we are trying to predict the time at which hyperedge event h occurs from the last event time for the nodes in the hyperedge t_h^p. This can be estimated using the condition probability density as $\hat{t_i} = \int_{t_h^p}^{\infty} t\mathrm{P}_h(t|\mathcal{E}(t_h^p))\mathrm{d}t$. Here, $\mathrm{P}_h(t|\mathcal{E}(t_h^p))$ is defined as in Equation 5.1, and the integral is computed using Monte-Carlo integration (Robert and Casella, 2004). Then for evaluation, we use Mean Absolute Error (MAE) metric, $MAE = \frac{1}{N}\sum_{i=1}^{N}|\hat{t_i} - t_i|$ for all the samples in the test.

5.5.3 Training Details

In all our experiments, we fixed the embedding dimension at $d = 64$, hidden and input dimension for the RNN at 64, batch size $B = 128$, number of noise streams $N^q = 20$, and number of corrupted hyperedges per true hyperedge is $N^e = 20$. The training is done for 200 epochs, and the model parameters that gave the least validation score $\mathcal{L}_{NCE} + \mathcal{L}^s_{NCE}$ are used for testing. For all the training, we use the AdamW optimizer (Reddi et al., 2018, Loshchilov and Hutter, 2019) of PyTorch (Kingma and Ba, 2015) with learning rate set to 0.0005. The first 50% of interactions is used for training, the next 25% for validation, and the rest for testing.

Methods	Enron		Twitter		Boston		Obama		Pope		Cannes	
	AUC	**MAE**	**AUC**	**MAE**	**AUC**	**MAE**	**AUC**	**MAE**	**AUC**	**MAE**	**AUC**	**MAE**
TGN	85.8 ± 2.9	NA	97.02 ± 0.5	NA	**89.4 ± 1.5**	NA	93.0 ± 1.0	NA	92.5 ± 1.2	NA	87.6 ± 2.4	NA
HyperTPP-j	91.6 ± 0.7	3.4 ± 0.0	97.2 ± 0.0	**30.0 ± 0.6**	85.4 ± 0.8	46.5 ± 0.7	90.7 ± 0.3	35.5 ± 1.1	85.2 ± 6.0	40.21 ± 1.2	90.7 ± 1.0	**24.0 ± 2.7**
HyperTPP-o	91.6 ± 0.5	3.7 ± 0.3	96.8 ± 0.4	30.8 ± 0.6	84.6 ± 1.2	51.8 ± 2.9	89.2 ± 0.5	38.6 ± 1.5	88.8 ± 1.4	37.3 ± 2.1	89.1 ± 1.6	26.8 ± 2.4
HyperTPP-f	90.8 ± 1.0	3.7 ± 0.0	97.3 ± 0.1	33.4 ± 0.5	83.4 ± 0.7	49.6 ± 2.1	86.6 ± 0.2	37.4 ± 1.5	91.8 ± 0.8	37.6 ± 0.2	86.4 ± 0.5	28.2 ± 2.1
RRHyperTPP(no-sup)-j	91.3 ± 1.3	3.5 ± 0.0	97.9 ± 0.1	33.2 ± 0.7	87.5 ± 3.3	47.8 ± 0.6	93.1 ± 0.2	33.4 ± 0.5	90.8 ± 0.5	38.2 ± 0.8	90.7 ± 0.2	27.0 ± 0.8
RRHyperTPP(no-sup)-o	91.8 ± 1.1	3.4 ± 0.2	97.7 ± 0.1	32.3 ± 0.2	87.5 ± 0.5	50.2 ± 1.1	91.1 ± 0.9	37.2 ± 2.5	89.3 ± 0.6	38.8 ± 0.3	90.3 ± 0.6	30.0 ± 0.9
RRHyperTPP(no-sup)-f	92.3 ± 0.1	3.6 ± 0.3	97.9 ± 0.1	33.4± 0.3	87.0 ± 0.2	**44.3 ± 0.9**	90.9 ± 0.3	32.3 ± 1.0	92.4 ± 0.3	36.9 ± 0.7	87.08 ± 0.7	26.9 ± 0.6
RRHyperTPP-j	93.3 ± 0.7	**3.3 ± 0.2**	**98.5 ± 0.1**	30.6 ± 1.8	89.2 ± 0.1	45.7 ± 0.4	**93.5 ± 0.5**	32.5 ± 0.4	**94.2 ± 0.4**	36.2 ± 1.0	**92.1 ± 0.4**	25.6 ±1.1
RRHyperTPP-o	**93.5 ± 0.2**	3.7 ± 0.1	98.3 ± 0.1	30.1 ± 0.9	88.0 ± 1.7	46.3 ± 3.1	92.3 ± 1.0	35.3 ± 2.0	94.0 ± 0.7	**35.4 ± 1.2**	91.2 ± 0.7	28.6 ± 0.6
RRHyperTPP-f	93.4 ± 0.3	3.5 ± 0.2	98.4 ± 0.1	32.3 ± 0.5	89.2 ± 0.2	45.1 ± 0.4	92.9 ± 0.3	**31.8 ± 0.3**	93.9 ± 0.3	37.4 ± 0.2	89.3 ± 0.4	25.2 ± 0.9

Table 5.3: Performance on forecasting in interaction type and interaction duration prediction tasks on depth one hyperedges. Here, interaction type prediction is evaluated using AUC in %, and interaction duration prediction is evaluated using MAE. The proposed model RRHyperTPP beats baseline models in almost in settings.

Methods	ICEWS-India		ICEWS-Nigeria	
	AUC	**MAE**	**AUC**	**MAE**
RRHyperTPP-j	58.7 ± 0.7	0.99 ± 0.00	**61.0 ± 0.38**	0.98 ± 0.00
RRHyperTPP-o	58.2 ± 1.4	0.99 ± 0.00	60.5 ± 0.50	**0.81 ± 0.13**
RRHyperTPP-f	**58.9 ± 0.2**	**0.75 ± 0.25**	58.7 ± 0.39	0.97 ± 0.00

Table 5.4: Performance on forecasting in interaction type and interaction duration prediction tasks on depth two hyperedges. Here, interaction type prediction is evaluated using AUC in %, and interaction duration prediction is evaluated using MAE.

5.5.4 Baselines

We use TGN (Rossi et al., 2020) to compare our model's performance against state-of-the-art pairwise interaction prediction models. Further, to show the advantage of the recursive hyperedge link prediction in Section 5.4, we created baseline model HyperTPP to forecast hyperedges created from the nodes involved in the interactions, $h = \{v_i; v_i \in h^0, (h^0, r^0) \in \ldots \in h\}$, and this model is trained using the same loss as RRHyperTPP. Additionally, we created RRHyperTPP (no-sup) to show the performance gain obtained when we use the supervised contrastive loss in Equation 5.11 along with the TPP based noise contrastive loss.

5.5.5 Results

Table 5.3 shows the performance of our models RRHyperTPP against baseline models. Here, the models that use time projection, ODEs, and time embeddings for **Drift** stage are denoted by -j, -o, and -f at the end, respectively. We can observe that forecasting performance with all three **Drift** stages is similar. However, theoretically, Neural ODEs should perform better than others as they have fewer assumptions when compared to Fourier and time projection-based methods. We have also done experiments on depth two hyperedges, and the results are shown in Table 5.4. Here, we can also observe that no single model outperforms the rest in all the tasks. Further, there is an average increment of 3.6% in the AUC metric for RRHyperTPP models when compared to HyperTPP based models in interaction type prediction tasks. For the interaction duration prediction task, we observed a reduction of 4.6% MAE when compared to HyperTPP models. Hence, we can conclude that modeling the recursive group structure of interaction resulted in performance improvement.

Moreover, we compare our model against TGN, a state-of-the-art pairwise interaction forecasting model. For the task of interaction type prediction, we can observe an increment of 2.9%, 2.3%, and 2.2% for models RRHyperTPP-j, RRHyperTPP-f, and RRHyperTPP-o, respectively. Hence, we can conclude that recursive hyperedge-based models perform better than

pairwise link prediction models.

Furthermore, we compare our models RRHyperTPP-j, RRHyperTPP-f, and RRHyperTPP-o against models RRHyperTPP(no-sup)-j, RRHyperTPP(no-sup)-f, and RRHyperTPP(no-sup)-o, respectively, to observe the performance gain obtained when we use the supervised contrastive loss in Equation 5.11 along with the TPP based noise contrastive loss. In the interaction type prediction task, our models RRHyperTPP-j, RRHyperTPP-f, and RRHyperTPP-o gave a performance improvement of 1.7% in the AUC metric over their respective baseline models. For interaction duration prediction, we observe that models RRHyperTPP-o and RRHyperTPP-f models have reduced MAE values by 3.3% and 8.2% over RRHyperTPP(no-sup)-j and RRHyperTPP(no-sup)-o, respectively. Hence, we can conclude that adding the supervised noise-contrastive loss improved the performance.

5.6 Summary

In this chapter, we have shown the significance of using the multi-relational recursive hyperedges for interaction models as it is a more natural approximation of real-world events. Here, we created a dynamic node representation and link predictor framework that can use this intricate relationship of interactions to forecast future events. In addition to this, we also created a noise contrastive learning framework that avoids the integration calculation in the survival function and can train when the number of types of events is of exponential order. To evaluate the model, we curated six datasets of depth one and two datasets of depth two hyperedges. We observed that our model RRHyperTPP is performing better previous state-of-the-art hyperedge event prediction model HyperTPP and temporal graph link prediction model TGN. As a future direction, we plan to explore new domains of application of these models and new architecture that uses higher-order information to learn efficient node representations. In addition to this, we also plan on improving the noise generation strategy by using historical data, so the noise generated samples will look more closely to actual data, thereby reducing the training iterations.

Chapter 6

Concluding Remarks

This thesis demonstrates the significance of modeling interaction as events on higher-order networks, as they offer a more accurate representation of real-world systems. In this thesis, we address the problem of forecasting higher-order edge formation events and provide solutions in various settings. The contributions made in this thesis are towards building efficient deep neural network-based approaches and training strategies for these models.

In the first contribution, we propose a TPP model to forecast events in a temporal hypergraph. For this, we propose a recurrent neural based architecture for dynamic representation for the nodes followed by a hyperedge prediction-based link predictor. The main challenge in training these models is that the calculation of likelihood is not computationally feasible as the number of possible hyperedges in a network of $|\mathcal{V}|$ nodes can be at most of $2^{|\mathcal{V}|}$ types and only a few of these hyperedges will be observed in the data, also, due to large number of points in training. To mitigate this, we propose mini-batch training with negative sampling to incorporate unseen hyperedges.

In our second contribution, we propose a learning strategy to forecast the events in a higher-order network in a scalable way. This is because the number of possible event types is exponential in the number of nodes, as mentioned earlier. Our strategy divides forecasting into multiple subtasks, which can be more easily and efficiently solved. In the first task, we predict the nodes that observe events in the immediate future using TPP based model. Then conditioned on the nodes and event time, we predict their neighborhood and the size of the hyperedges these nodes are involved in. Then candidate hyperedges can be generated based on these to hyperedge link prediction, which can identify the true hyperedges from false hyperedge. Here, we also proposed a scalable, dynamic node embedding module using a memory module. Here, instead of performing the sequential operation of updating the node embeddings after each interaction as done in Section 3.1.3. We propose a technique of storing the interaction features of each node

in the batch and then updating the node embeddings before the start of the next batch. This allows one to do batch data processing for faster training. Further, we also use the temporal graph attention layer to extract recent information from its recent temporal neighbors. We evaluated this model in forecasting events in a directed temporal hypergraph created from five open-source data. In addition to this, we also proposed a directed hyperedge link prediction module that models all the types of relations in a directed hyperedge.

In our final contribution, we address the problem of modeling the grouping structure within an interaction. This is done using multi-relational recursive hyperedges, where a hyperedge can act as a node in another hyperedge. Also, here each hyperedge is associated with a relation. In addition, we devise a noise contrastive learning strategy to train the models by avoiding the calculation of computationally expensive survival functions involved in the TPP likelihood. To evaluate this model, we developed eight temporal multi-relational recursive hypergraphs from open-source data.

Future Work. One crucial direction to consider for future work is modeling processes occurring on different network topological structures and tailoring the modeling approach to the specific nature of the process we aim to study. For instance, an epidemic progression in populations or a virus within a computer network can be effectively modeled as events happening on the nodes. Research in higher-order dynamical systems is dedicated to modeling the evolution of such phenomena (Bick et al., 2023). However, it is important to note that Bick et al. (2023) assumes that higher-order networks are static. Incorporating the temporal nature of edge formation in these networks would enable one to build models that offer closer approximations to real-world processes. For instance, a contact network in epidemic progression can be modeled as a higher-order network with nodes representing people. These dynamic models can be applied to various practical applications such as contact tracing or identifying the zeroth patient, potentially leading to more accurate and effective disease control strategies.

Furthermore, there is considerable room for improvement in the architecture of the encoder and decoder modules. While Chapters 3 and 4 predominantly utilized hypergraph structures, our research in Chapter 5 highlighted the importance of moving beyond hypergraphs to capture higher-order interactions effectively. The emerging topological deep learning field addresses some of these fields in static higher-order networks by introducing new network structure for modeling higher-order information (Horn et al., 2022, Hajij et al., 2023). Similarly, in the domain of temporal networks, research is being conducted to improve temporal graph neural networks by better extracting information from their interaction history by using temporal multi-hop neighborhood information and higher-order temporal motif information (Wang et al., 2021, Jin et al., 2022). Another important area on this is to create interpretable models that

will improve the trustworthiness of the forecast. Recent work by Xia et al. (2023) explores these ideas for link prediction problems in a pairwise temporal graph using a reinforcement learning framework. Similarly, efforts are being made to explore counterfactual scenarios to explain forecasts in both TPP (Noorbakhsh and Rodriguez, 2022) and graph neural network domains (Lucic et al., 2022). By incorporating these ideas, one can improve the accuracy and reliability of the existing models.

In Chapter 4, we explored methods to simulate hyperedges through a multi-task approach and provided a scalable architecture for faster training. An alternate direction to achieve scalability is creating better data structures for feature computing and doing parallel processing of the samples. TGL (Zhou et al., 2022) explores this by training temporal graph neural networks on graphs with billions of edges through parallel GPU computation promoting data structures and neighborhood sampling techniques. NAT (Luo and Li, 2022) introduces the N-cache data structure to store the interaction neighborhood of each node, enabling faster structural feature calculation. Exploring these ideas for higher-order interactions will enhance the models' applicability in real-world problems, such as recommendation systems dealing with a large number of entities.

Furthermore, drawing from the success of data science challenges in natural language processing and computer vision, where leaderboards have driven the development of state-of-the-art models like BERT (Devlin et al., 2019), GPT (Brown et al., 2020) and ResNet (He et al., 2016), it becomes evident that promoting research in the higher-order network domain necessitates high-quality open datasets and a standardized benchmark framework with multiple evaluation criteria. The recently developed Temporal Graph Benchmark (Huang et al., 2023) have created a framework for this using a common data loader and evaluation metrics for pairwise graphs. However, this framework focuses on models that use temporal link prediction loss and employs node classification and link prediction evaluation metrics. There is a need to create a similar framework for event prediction in higher-order graphs with time prediction and interaction-type tasks. Additionally, exploring other higher-order interaction properties, such as simplicial closures, reciprocity, and recurrences, would significantly enrich the evaluation process and lead to more comprehensive and practical models.

Outlook. Study of networks has been central to machine learning for many decades. Many advancements in this area have been based on representation learning and graph neural networks to study problems like node classification, edge prediction, and community detection in a network. Only very recently, similar problems have also been studied in higher-order networks that consider multiple-entity interactions. To consider the temporal nature of real-world networks, earlier studies have devised techniques that model the dynamics of the evolution of

the networks while being able to address tasks like node classification, link prediction, and community detection. A similar study in the domain of higher-order of networks poses very challenging problems, and these problems have not been studied so far. Through this thesis, we have introduced several techniques to forecast edge formation events in these higher-order networks. We believe that this will trigger new research and outlook into this domain.

Bibliography

Abbeel, P. and A. Y. Ng (2004). Apprenticeship learning via inverse reinforcement learning. In *International Conference on Machine Learning*, pp. 1. 32

Adiga, A., S. Athreya, K. R. Bhimala, A. Dukkipati, T. Gracious, S. Gupta, B. Hurt, G. Kaur, B. Lewis, M. Marathe, G. K. Patra, N. Rathod, R. Sundaresan, S. Venkataramanan, and S. Yasodharan (2023). A multi-team multi-model collaborative covid-19 forecasting hub for india. In *Proceedings of Winter Simulation Conference (WSC)*. 37

Arastuie, M., S. Paul, and K. Xu (2020). Chip: a hawkes process model for continuous-time networks with scalable and consistent estimation. *Advances in Neural Information Processing Systems 33*, 16983–16996. 5, 36

Arjovsky, M., S. Chintala, and L. Bottou (2017). Wasserstein generative adversarial networks. In *International Conference on Machine Learning*, Volume 70, pp. 214–223. 30, 31

Baddeley, A., I. Bárány, and R. Schneider (2007). *Stochastic Geometry*. Springer Berlin Heidelberg. 18

Bai, S., F. Zhang, and P. H. Torr (2021). Hypergraph convolution and hypergraph attention. *Pattern Recognition 110*, 107637. 50

Ben Hassen, T. and H. El Bilali (2022). Impacts of the russia-ukraine war on global food security: Towards more sustainable and resilient food systems? *Foods 11*, 15. 1

Benson, A. R., R. Abebe, M. T. Schaub, A. Jadbabaie, and J. Kleinberg (2018). Simplicial closure and higher-order link prediction. *Proceedings of the National Academy of Sciences 115*, E11221–E11230. 9, 53

Benson, A. R., R. Kumar, and A. Tomkins (2018). Sequences of sets. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 1148–1157. 9

Bick, C., E. Gross, H. A. Harrington, and M. T. Schaub (2023). What are higher-order networks? *SIAM Review*. 104

Blundell, C., J. Beck, and K. A. Heller (2012). Modelling reciprocating relationships with hawkes processes. In *Advances in Neural Information Processing Systems*, Volume 25, pp. 2600–2608. 36

Bordes, A., N. Usunier, A. Garcia-Duran, J. Weston, and O. Yakhnenko (2013). Translating embeddings for modeling multi-relational data. In *Advances in Neural Information Processing Systems*, Volume 26. 7

Boschee, E., J. Lautenschlager, S. O'Brien, S. Shellman, J. Starz, and M. Ward (2015). Icews coded event data. 98

Box, G. E. P. and G. Jenkins (1990). *Time series analysis, forecasting, and control*. Holden-Day, Inc. 2

Brown, E. N., R. Barbieri, V. Ventura, R. E. Kass, and L. M. Frank (2002, 02). The time-rescaling theorem and its application to neural spike train data analysis. *Neural Computation 14*(2), 325–346. 28, 33

Brown, T., B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, et al. (2020). Language models are few-shot learners. In *Advances in neural information processing systems*, Volume 33, pp. 1877–1901. 105

Cao, J., X. Lin, X. Cong, S. Guo, H. Tang, T. Liu, and B. Wang (2021). Deep structural point process for learning temporal interaction networks. In *Machine Learning and Knowledge Discovery in Databases. Research Track*, pp. 305–320. 6, 37, 45, 50

Chen, R. T. Q., B. Amos, and M. Nickel (2021). Neural spatio-temporal point processes. In *International Conference on Learning Representations*. 27

Chen, R. T. Q., Y. Rubanova, J. Bettencourt, and D. K. Duvenaud (2018). Neural ordinary differential equations. In *Advances in Neural Information Processing Systems*, Volume 31. 27, 88, 97

Chen, Y. (2016). Thinning algorithms for simulating point processes. 15, 93

Cho, K., B. van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio (2014, October). Learning phrase representations using RNN encoder–decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods*

in Natural Language Processing (EMNLP), pp. 1724–1734. Association for Computational Linguistics. 75

Chodrow, P. and A. Mellor (2019). Annotated hypergraphs: Models and applications. *Applied Network Science 5*, 1–25. 77, 98

Choo, H. and K. Shin (2022). On the persistence of higher-order interactions in real-world hypergraphs. In *Proceedings of the 2022 SIAM International Conference on Data Mining (SDM)*, pp. 163–171. SIAM. 9

Comrie, C. and J. Kleinberg (2021). Hypergraph ego-networks and their temporal evolution. In *IEEE International Conference on Data Mining (ICDM)*, pp. 91–100. 9

Cox, D. R. (1955). Some statistical methods connected with series of events. *Journal of the Royal Statistical Society: Series B (Methodological) 17*(2), 129–157. 21

Cramér, H. (1969). Historical review of filip lundberg's works on risk theory. *Scandinavian Actuarial Journal 1969*, 6–12. 4

da Xu, chuanwei ruan, evren korpeoglu, sushant kumar, and kannan achan (2020). Inductive representation learning on temporal graphs. In *International Conference on Learning Representations*. ix, 6, 7, 50, 56, 57, 60, 75, 85, 88, 96

Dai, H., Y. Wang, R. Trivedi, and L. Song (2016). Deep coevolutionary network: Embedding user and item features for recommendation. 5, 45, 47, 51, 55

Daley, D. J. and D. Vere-Jones (2003). *An introduction to the theory of point processes: Volume I: Elementary theory and methods*. Springer New York, NY. 3, 18

Daley, D. J. and D. Vere-Jones (2007). *An introduction to the theory of point processes: Volume II: General theory and structure*. Springer New York, NY. 18

Dasgupta, S. S., S. N. Ray, and P. P. Talukdar (2018). HyTE: Hyperplane-based temporally aware knowledge graph embedding. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pp. 2001–2011. 8

Devlin, J., M.-W. Chang, K. Lee, and K. Toutanova (2019, June). BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pp. 4171–4186. 105

Devroye, L. (1986). *Non-Uniform Random Variate Generation*. Springer New York, NY. 32

Domenico, M. D. and E. G. Altmann (2020). Unraveling the origin of social bursts in collective attention. *Scientific Reports 10*, 4629. 98

Du, N., H. Dai, R. Trivedi, U. Upadhyay, M. Gomez-Rodriguez, and L. Song (2016). Recurrent marked temporal point processes: Embedding event history to vector. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 1555–1564. 4, 26, 37

DuBois, C., C. Butts, and P. Smyth (2013). Stochastic blockmodeling of relational event dynamics. In *Proceedings of the Sixteenth International Conference on Artificial Intelligence and Statistics*, Volume 31, pp. 238–246. 5

Dukkipati, A., T. Gracious, and S. Gupta (2021). Covihawkes: Temporal point process and deep learning based covid-19 forecasting for india. 37

Durkan, C., A. Bekasov, I. Murray, and G. Papamakarios (2019). Neural spline flows. In *Advances in Neural Information Processing Systems*, Volume 32. 28

Ertekin, S., C. Rudin, and T. H. McCormick (2015). Reactive point processes: A new approach to predicting power failures in underground electrical systems. *The Annals of Applied Statistics 9*(1), 122–144. 21

Fatemi, B., P. Taslakian, D. Vazquez, and D. Poole (2020, 7). Knowledge hypergraphs: Prediction beyond binary relations. In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 2191–2197. 64, 86

Fawcett, T. (2006). An introduction to roc analysis. *Pattern Recognition Letters 27*(8), 861–874. 84, 99

Filimonov, V. and D. Sornette (2012). Quantifying reflexivity in financial markets: Toward a prediction of flash crashes. *Phys. Rev. E 85*, 056108. 4

Fowler, J. H. (2006). Legislative cosponsorship networks in the us house and senate. *Social Networks 28*(4), 454–465. 54

Frühwirth-Schnatter, S. (2006). *Finite mixture and Markov switching models*, Volume 425. Springer. 29, 32

Garcia-Duran, A., S. Dumančić, and M. Niepert (2018). Learning sequence encoders for temporal knowledge graph completion. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pp. 4816–4821. 8

Gehrke, J., P. Ginsparg, and J. Kleinberg (2003, 12). Overview of the 2003 kdd cup. *SIGKDD Explor. Newsl. 5*(2), 149–151. 78

Ghosh, J. K. (2009). Survival and event history analysis: A process point of view by odd o. aalen, Ørnulf borgan, håkon k. gjessing. *International Statistical Review 77*(3), 463–464. 6, 8, 21

Ghoshdastidar, D. and A. Dukkipati (2017a). Consistency of spectral hypergraph partitioning under planted partition model. *The Annals of Statistics 45*(1), 289–315. 45

Ghoshdastidar, D. and A. Dukkipati (2017b). Uniform hypergraph partitioning: Provable tensor methods and sampling techniques. *The Journal of Machine Learning Research 18*(50), 1–41. 45

Goodfellow, I., J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio (2014). Generative adversarial nets. In *Advances in Neural Information Processing Systems*, Volume 27. 30

Goyal, P., S. R. Chhetri, and A. Canedo (2020). Dyngraph2vec: Capturing network dynamics using dynamic graph representation learning. *Knowledge-Based Systems 187*, 104816. 6

Goyal, P., N. Kamra, X. He, and Y. Liu (2018). Dyngem: Deep embedding method for dynamic graphs. 6

Gracious, T. and A. Dukkipati (2023). Dynamic representation learning with temporal point processes for higher-order interaction forecasting. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Volume 37, pp. 7748–7756. 45

Gracious, T., A. Gupta, and A. Dukkipati (2023). Higher order and directional network event forecasting using temporal point process. Submitted. 66

Gracious, T., S. Gupta, A. Kanthali, R. M. Castro, and A. Dukkipati (2021). Neural latent space model for dynamic networks and temporal knowledge graphs. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Volume 35, pp. 4054–4062. 6, 85

Grathwohl, W., R. T. Chen, J. Betterncourt, I. Sutskever, and D. Duvenaud (2019). Ffjord: Free-form continuous dynamics for scalable reversible generative models. In *International Conference on Learning Representations*. 29

Guo, R., J. Li, and H. Liu (2018, 7). Initiator: Noise-contrastive estimation for marked temporal point process. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18*, pp. 2191–2197. 31

Gupta, A., M. Farajtabar, B. Dilkina, and H. Zha (2018). Discrete interventions in hawkes processes with applications in invasive species management. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence*, pp. 3385–3392. 32

Gutmann, M. and A. Hyvärinen (2010). Noise-contrastive estimation: A new estimation principle for unnormalized statistical models. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, Volume 9, pp. 297–304. 31

Hairer, E., S. P. Nørsett, and G. Wanner (1993). *Solving Ordinary Differential Equations I (2nd Revised. Ed.): Nonstiff Problems*. Berlin, Heidelberg: Springer-Verlag. 27

Hajij, M., G. Zamzmi, T. Papamarkou, N. Miolane, A. Guzmán-Sáenz, K. N. Ramamurthy, T. Birdal, T. K. Dey, S. Mukherjee, S. N. Samaga, et al. (2023). Topological deep learning: Going beyond graph data. 86, 104

Han, Z., Y. Ma, Y. Wang, S. Günnemann, and V. Tresp (2020). Graph hawkes neural network for forecasting on temporal knowledge graphs. In *Automated Knowledge Base Construction*. 8, 37

Hawkes, A. G. (1971). Spectra of some self-exciting and mutually exciting point processes. *Biometrika 58*(1), 83–90. 4, 23

Hawkes, A. G. and D. Oakes (1974). A cluster process representation of a self-exciting process. *Journal of Applied Probability 11*(3), 493–503. 23

He, K., X. Zhang, S. Ren, and J. Sun (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778. 105

Ho, J., X. Chen, A. Srinivas, Y. Duan, and P. Abbeel (2019). Flow++: Improving flow-based generative models with variational dequantization and architecture design. In *International Conference on Machine Learning*, Volume 97, pp. 2722–2730. 33

Ho, J., A. Jain, and P. Abbeel (2020). Denoising diffusion probabilistic models. In *Advances in Neural Information Processing Systems*, Volume 33, pp. 6840–6851. 29

Hochreiter, S. and J. Schmidhuber (1997). Long short-term memory. *Neural Computation 9*(8), 1735–1780. 2, 6, 38

Horn, M., E. D. Brouwer, M. Moor, Y. Moreau, B. Rieck, and K. Borgwardt (2022). Topological graph neural networks. In *International Conference on Learning Representations*. 86, 104

Huang, C.-W., D. Krueger, A. Lacoste, and A. Courville (2018). Neural autoregressive flows. In *International Conference on Machine Learning*, Volume 80, pp. 2078–2087. 29

Huang, S., F. Poursafaei, J. Danovitch, M. Fey, W. Hu, E. Rossi, J. Leskovec, M. Bronstein, G. Rabusseau, and R. Rabbany (2023). Temporal graph benchmark for machine learning on temporal graphs. 105

Huang, Z., H. Soliman, S. Paul, and K. S. Xu (2022). A mutually exciting latent space hawkes process model for continuous-time networks. In *The 38th Conference on Uncertainty in Artificial Intelligence*. 5, 36

Hwang, H., S. Lee, C. Park, and K. Shin (2022). Ahp: Learning to negative sample for hyperedge prediction. In *Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 2237–2242. 85

Isham, V. and M. Westcott (1979). A self-correcting point process. *Stochastic Processes and their Applications 8*(3), 335–347. 23

Jacobsen, M. (2006, 01). *Point Process Theory and Applications: Marked Point and Piecewise Deterministic Processes*. Springer Science & Business Media. 18

Jaini, P., K. A. Selby, and Y. Yu (2019). Sum-of-squares polynomial flow. In *International Conference on Machine Learning*, Volume 97, pp. 3009–3018. 28, 29

Jia, J. and A. R. Benson (2019). Neural jump stochastic differential equations. In *Advances in Neural Information Processing Systems*, Volume 32. 27

Jin, M., Y.-F. Li, and S. Pan (2022). Neural temporal walks: Motif-aware representation learning on continuous-time dynamic graphs. In *Advances in Neural Information Processing Systems*. 104

Jin, W., C. Coley, R. Barzilay, and T. Jaakkola (2017). Predicting organic reaction outcomes with weisfeiler-lehman network. In *Advances in Neural Information Processing Systems*, Volume 30. 78

Jin, W., M. Qu, X. Jin, and X. Ren (2020). Recurrent event network: Autoregressive structure inference over temporal knowledge graphs. In *EMNLP*. 8

Junuthula, R., M. Haghdan, K. S. Xu, and V. Devabhaktuni (2019). The block point process model for continuous-time event-based dynamic networks. In *The World Wide Web Conference*, pp. 829–839. 5, 36

Karimi, M. R., E. Tavakoli, M. Farajtabar, L. Song, and M. Gomez Rodriguez (2016). Smart broadcasting: Do you want to be seen? In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 1635–1644. 32

Kazemi, S. M., R. Goel, K. Jain, I. Kobyzev, A. Sethi, P. Forsyth, and P. Poupart (2020). Representation learning for dynamic graphs: A survey. *The Journal of Machine Learning Research 21*(1), 2648–2720. 59, 75, 96

Kazemi, S. M. and D. Poole (2018). Simple embedding for link prediction in knowledge graphs. In *Advances in Neural Information Processing Systems*, Volume 31. 7

Kidger, P., J. Morrill, J. Foster, and T. Lyons (2020). Neural controlled differential equations for irregular time series. In *Advances in Neural Information Processing Systems*, Volume 33. 27

Kim, S., M. Choe, J. Yoo, and K. Shin (2022, 12). Reciprocity in directed hypergraphs: Measures, findings, and generators. In *2022 IEEE International Conference on Data Mining (ICDM)*, pp. 1005–1010. 9

King, Z. A., J. Lu, A. Dräger, P. Miller, S. Federowicz, J. A. Lerman, A. Ebrahim, B. O. Palsson, and N. E. Lewis (2015, 10). Bigg models: A platform for integrating, standardizing and sharing genome-scale models. *Nucleic Acids Research 44*(D1), D515–D522. 78

Kingma, D. P. and J. Ba (2015). Adam: A method for stochastic optimization. In *International Conference on Learning Representations*. 58, 99

Klimt, B. and Y. Yang (2004). The enron corpus: A new dataset for email classification research. In *European Conference on Machine Learning*, pp. 217–226. 53

Kobyzev, I., S. J. Prince, and M. A. Brubaker (2020). Normalizing flows: An introduction and review of current methods. *IEEE transactions on pattern analysis and machine intelligence 43*(11), 3964–3979. 29

Kumar, S., X. Zhang, and J. Leskovec (2019). Predicting dynamic embedding trajectory in temporal interaction networks. In *Proceedings of the 25th ACM SIGKDD international conference on Knowledge discovery and data mining*. 7, 64, 85, 87, 96

Kutta, W. (1901). Beitrag zur näherungsweisen Integration totaler Differentialgleichungen. *Zeitschrift für Mathematik und Physik 46*, 435–53. 27

Lee, D. and K. Shin (2023a). I'm me, we're us, and i'm us: Tri-directional contrastive learning on hypergraphs. In *Proceedings of the AAAI Conference on Artificial Intelligence*. 65

Lee, G. and K. Shin (2023b, 2). Temporal hypergraph motifs. *Knowl. Inf. Syst. 65*(4), 1549–1586. 9

Lewis, E. and G. Mohler (2011). A nonparametric em algorithm for multiscale hawkes processes. *Journal of nonparametric statistics 1*(1), 1–20. 4, 23, 30

Lewis, E., G. Mohler, P. J. Brantingham, and A. Bertozzi (2011). Self-exciting point process of insurgency in iraq. *Security Journal 25*, 0955–1662. 4

Lewis, P. A. W. and G. S. Shedler (1979). Simulation of nonhomogeneous poisson processes by thinning. *Naval Research Logistics Quarterly 26*(3), 403–413. 4, 34

Li, S., S. Xiao, S. Zhu, N. Du, Y. Xie, and L. Song (2018). Learning temporal point processes via reinforcement learning. In *Advances in Neural Information Processing Systems*, Volume 31. 31

Lin, H., C. Tan, L. Wu, Z. Gao, and S. Z. Li (2021). An empirical study: Extensive deep temporal point process. 15

Lin, H., L. Wu, G. Zhao, L. Pai, and S. Z. Li (2022). Exploring generative neural temporal point process. *Transactions on Machine Learning Research*. 4, 15, 29, 33

Liu, Y., J. Ma, and P. Li (2022). Neural predicting higher-order patterns in temporal networks. In *Proceedings of the ACM Web Conference 2022*, WWW '22, pp. 1340–1351. 9

Liu, Z., X. Xie, and L. Chen (2018). Context-aware academic collaborator recommendation. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 1870–1879. 65

Loshchilov, I. and F. Hutter (2019). Decoupled weight decay regularization. In *International Conference on Learning Representations*. 99

Lucic, A., M. A. Ter Hoeve, G. Tolomei, M. De Rijke, and F. Silvestri (2022). Cf-gnnexplainer: Counterfactual explanations for graph neural networks. In *Proceedings of The 25th International Conference on Artificial Intelligence and Statistics*, Volume 151, pp. 4499–4511. 105

Luo, Y. and P. Li (2022). Neighborhood-aware scalable temporal network representation learning. In *Proceedings of the First Learning on Graphs Conference*, Volume 198. 105

Matias, C., T. Rebafka, and F. Villers (2018). A semiparametric extension of the stochastic block model for longitudinal networks. *Biometrika 105*(3), 665–680. 5

Mehrasa, N., R. Deng, M. O. Ahmed, B. Chang, J. He, T. Durand, M. Brubaker, and G. Mori (2019). Point process flows. 33

Mei, H. (2021). *Neural Probabilistic Methods for Event Sequence Modeling*. Ph. D. thesis, Johns Hopkins University. 15

Mei, H. and J. M. Eisner (2017). The neural hawkes process: A neurally self-modulating multivariate point process. In *Advances in Neural Information Processing Systems*, Volume 30. 4, 26

Mei, H., T. Wan, and J. Eisner (2020). Noise-contrastive estimation for multivariate point processes. *Advances in Neural Information Processing Systems 33*, 5204–5214. 31, 91, 93

Møller, J. and J. G. Rasmussen (2005). Perfect simulation of hawkes processes. *Advances in applied probability 37*(3), 629–646. 32

Møller, J. and J. G. Rasmussen (2006). Approximate simulation of hawkes processes. *Methodology and Computing in Applied Probability 8*, 53–64. 32

Moller, J. and R. P. Waagepetersen (2003). *Statistical inference and simulation for spatial point processes*. CRC press. 18

Ng, A. Y. and S. J. Russell (2000). Algorithms for inverse reinforcement learning. In *International Conference on Machine Learning*, pp. 663–670. 32

Noorbakhsh, K. and M. G. Rodriguez (2022). Counterfactual temporal point processes. In *Advances in Neural Information Processing Systems*. 105

Ogata, Y. (1981). On lewis' simulation method for point processes. *IEEE Transactions on Information Theory 27*(1), 23–31. 4, 34

Ogata, Y. (1988). Statistical models for earthquake occurrences and residual analysis for point processes. *Journal of the American Statistical association 83*(401), 9–27. 18

Ogata, Y. (1998). Space-time point-process models for earthquake occurrences. *Annals of the Institute of Statistical Mathematics 50*(2), 379–402. 4

Omi, T., Y. Ogata, Y. Hirata, and K. Aihara (2014). Estimating the etas model from an early aftershock sequence. *Geophysical Research Letters 41*(3), 850–857. 18

Omi, T., n. ueda, and K. Aihara (2019). Fully neural network based model for general temporal point processes. In *Advances in Neural Information Processing Systems*, Volume 32. 28

Omodei, E., M. De Domenico, and A. Arenas (2015). Characterizing interactions in online social networks during exceptional events. *Frontiers in Physics 3*, 59. 98

Paranjape, A., A. R. Benson, and J. Leskovec (2017). Motifs in temporal networks. In *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining*, pp. 601–610. 53

Park, N., F. Liu, P. Mehta, D. Cristofor, C. Faloutsos, and Y. Dong (2022). Evokg: Jointly modeling event time and network structure for reasoning over temporal knowledge graphs. In *Proceedings of the Fifteenth ACM International Conference on Web Search and Data Mining*, pp. 794–803. 8

Passino, F. S. and N. A. Heard (2022). Mutually exciting point process graphs for modeling dynamic networks. *Journal of Computational and Graphical Statistics 32*, 116–130. 5, 36

Paszke, A., S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala (2019). Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*, Volume 32. 58

Pemantle, R. (2007). A survey of random processes with reinforcement. *Probability Surveys 4*, 1–79. 21

Pontryagin, L. S. (1987). *Mathematical theory of optimal processes*. CRC press. 27

Qu, C., X. Tan, S. Xue, X. Shi, J. Zhang, and H. Mei (2022). Bellman meets hawkes: Model-based reinforcement learning via temporal point processes. 32

Ranshous, S., C. A. Joslyn, S. Kreyling, K. Nowak, N. F. Samatova, C. L. West, and S. Winters (2017). Exchange pattern mining in the bitcoin transaction directed hypergraph. In *Financial Cryptography and Data Security*, pp. 248–263. 65

Rasmussen, J. G. (2013). Bayesian inference for hawkes processes. *Methodology and Computing in Applied Probability 15*, 623–642. 30

Rasmussen, J. G. (2018). Lecture notes: Temporal point processes and the conditional intensity function. 15

Reddi, S. J., S. Kale, and S. Kumar (2018). On the convergence of adam and beyond. In *International Conference on Learning Representations*. 99

Rizoiu, M.-A., Y. Lee, S. Mishra, and L. Xie (2017). *Hawkes Processes for Events in Social Media*, pp. 191–218. Association for Computing Machinery and Morgan & Claypool. 24

Rizoiu, M.-A., S. Mishra, Q. Kong, M. Carman, and L. Xie (2018). Sir-hawkes: Linking epidemic models and hawkes processes to model diffusions in finite populations. In *Proceedings of the 2018 world wide web conference*, pp. 419–428. 38

Robert, C. and G. Casella (2004). *Monte Carlo Statistical Method*. Springer New York, NY. 26, 99

Rodriguez, M. G. and I. Valera (2018). Learning with temporal point processes. 15

Rossi, E., B. Chamberlain, F. Frasca, D. Eynard, F. Monti, and M. Bronstein (2020). Temporal graph networks for deep learning on dynamic graphs. ix, 7, 56, 57, 60, 66, 78, 101

Runge, C. (1895). Ueber die numerische auflösung von differentialgleichungen. *Mathematische Annalen 46*, 167–178. 27

S. Gupta, G. S. and A. Dukkipati (2019). A generative model for dynamic networks with applications. In *Proceedings of the AAAI Conference on Artificial Intelligence*. 65

Sankar, A., Y. Wu, L. Gou, W. Zhang, and H. Yang (2020). Dysat: Deep neural representation learning on dynamic graphs via self-attention networks. In *Proceedings of the 13th International Conference on Web Search and Data Mining*, pp. 519–527. 6

Sarkar, P. and A. W. Moore (2005). Dynamic social network analysis using latent space models. *Acm sigkdd explorations newsletter* 7(2), 31–40. 6

Schlichtkrull, M., T. N. Kipf, P. Bloem, R. Van Den Berg, I. Titov, and M. Welling (2018). Modeling relational data with graph convolutional networks. In *The Semantic Web*, pp. 593–607. 7

Shchur, O. (2022). *Modeling Continuous-time Event Data with Neural Temporal Point Processes*. Ph. D. thesis, Universität München. 15

Shchur, O., M. Biloš, and S. Günnemann (2020). Intensity-free learning of temporal point processes. In *International Conference on Learning Representations*. 28, 29, 33

Shchur, O., N. Gao, M. Biloš, and S. Günnemann (2020). Fast and flexible temporal point processes with triangular maps. In *Advances in Neural Information Processing Systems*. 28

Shchur, O., A. C. Türkmen, T. Januschowski, and S. Günnemann (2021). Neural temporal point processes: A review. In *IJCAI 2021*. 15

Silverman, B. W. (1986). *Density estimation for statistics and data analysis*, Volume 26. CRC press. 93

Soliman, H., L. Zhao, Z. Huang, S. Paul, and K. S. Xu (2022). The multivariate community hawkes model for dependent relational events in continuous-time networks. In *International Conference on Machine Learning*, pp. 20329–20346. 5, 36

Sun, Z., Z.-H. Deng, J.-Y. Nie, and J. Tang (2019). Rotate: Knowledge graph embedding by relational rotation in complex space. In *International Conference on Learning Representations*. 7

Teru, K., E. Denis, and W. Hamilton (2020). Inductive relation prediction by subgraph reasoning. In *International Conference on Machine Learning*, pp. 9448–9457. 7

Trivedi, R., H. Dai, Y. Wang, and L. Song (2017). Know-evolve: Deep temporal reasoning for dynamic knowledge graphs. In *International Conference on Machine Learning*, pp. 3462–3471. 8, 56

Trivedi, R., M. Farajtabar, P. Biswal, and H. Zha (2019). Dyrep: Learning representations over dynamic graphs. In *International Conference on Learning Representations*. ix, 6, 37, 45, 56, 60, 66

Trouillon, T., J. Welbl, S. Riedel, É. Gaussier, and G. Bouchard (2016). Complex embeddings for simple link prediction. In *International Conference on Machine Learning*, pp. 2071–2080. 7

Upadhyay, U., A. De, and M. Gomez Rodriguez (2018). Deep reinforcement learning of marked temporal point processes. In *Advances in Neural Information Processing Systems*, Volume 31. 32

Vashishth, S., S. Sanyal, V. Nitin, and P. Talukdar (2020). Composition-based multi-relational graph convolutional networks. In *International Conference on Learning Representations*. 7

Vaswani, A., N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin (2017). Attention is all you need. In *Advances in Neural Information Processing Systems*, Volume 30. 3, 6, 76, 97

Wang, Q., Z. Mao, B. Wang, and L. Guo (2017). Knowledge graph embedding: A survey of approaches and applications. *IEEE Transactions on Knowledge and Data Engineering 29*(12), 2724–2743. 7

Wang, Y., Y.-Y. Chang, Y. Liu, J. Leskovec, and P. Li (2021). Inductive representation learning in temporal networks via causal anonymous walks. In *International Conference on Learning Representations*. 104

Wang, Z., J. Zhang, J. Feng, and Z. Chen (2014). Knowledge graph embedding by translating on hyperplanes. In *Proceedings of the AAAI conference on artificial intelligence*, Volume 28. 7

Wei, T., Y. You, T. Chen, Y. Shen, J. He, and Z. Wang (2022). Augmentations in hypergraph contrastive learning: Fabricated and generative. In *Advances in Neural Information Processing Systems*. 65

Wen, J., J. Li, Y. Mao, S. Chen, and R. Zhang (2016). On the representation and embedding of knowledge bases beyond binary relations. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence*, pp. 1300–1307. 86

Xia, W., M. Lai, C. Shan, Y. Zhang, X. Dai, X. Li, and D. Li (2023). Explaining temporal graph models through an explorer-navigator framework. In *International Conference on Learning Representations*. 105

Xia, W., Y. Li, and S. Li (2022). Graph neural point process for temporal interaction prediction. *IEEE Transactions on Knowledge and Data Engineering 35*, 4867–4879. 6, 37

Xiao, S., M. Farajtabar, X. Ye, J. Yan, L. Song, and H. Zha (2017). Wasserstein learning of deep generative point process models. In *Advances in Neural Information Processing Systems*, Volume 30. 31

Xiao, S., J. Yan, X. Yang, H. Zha, and S. M. Chu (2017). Modeling the intensity function of point process via recurrent neural networks. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*, Volume 7, pp. 1597–1603. 28

Xu, H., W. Wu, S. Nemati, and H. Zha (2017). Patient flow prediction via discriminative learning of mutually-correcting processes. In *2017 IEEE 33rd International Conference on Data Engineering (ICDE)*, pp. 37–38. 23

Yan, J. (2019). Recent advance in temporal point process: from machine learning perspective. Technical report, Shanghai Jiao Tong University. 15

Yan, J., X. Liu, L. Shi, C. Li, and H. Zha (2018). Improving maximum likelihood estimation of temporal point process via discriminative and adversarial learning. In *IJCAI*, pp. 2948–2954. 31

Yang, B., W.-t. Yih, X. He, J. Gao, and L. Deng (2015). Embedding entities and relations for learning and inference in knowledge bases. In *International Conference on Learning Representations*. 7

Yang, J., V. A. Rao, and J. Neville (2017). Decoupling homophily and reciprocity with latent space network models. In *The Conference on Uncertainty in Artificial Intelligence*. 5, 36

Yin, H., A. R. Benson, J. Leskovec, and D. F. Gleich (2017). Local higher-order graph clustering. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 77

Zarezade, A., U. Upadhyay, H. R. Rabiee, and M. Gomez-Rodriguez (2017). Redqueen: An online algorithm for smart broadcasting in social networks. In *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining*, pp. 51–60. 32

Zhang, Q., A. Lipani, O. Kirnap, and E. Yilmaz (2020). Self-attentive hawkes process. In *Proceedings of the 37th International Conference on Machine Learning*, pp. 11183–11193. 4, 26

Zhang, R., C. Walder, and M.-A. Rizoiu (2020). Variational inference for sparse gaussian process modulated hawkes process. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Volume 34, pp. 6803–6810. 23

Zhang, R., Y. Zou, and J. Ma (2019). Hyper-sagnn: a self-attention based graph neural network for hypergraphs. In *International Conference on Learning Representations*. 47, 65, 78, 84

Zhou, F., Z. Li, X. Fan, Y. Wang, A. Sowmya, and F. Chen (2020). Efficient inference for nonparametric hawkes processes using auxiliary latent variables. *The Journal of Machine Learning Research 21*(1), 9745–9775. 23

Zhou, H., D. Zheng, I. Nisa, V. Ioannidis, X. Song, and G. Karypis (2022). Tgl: A general framework for temporal gnn training on billion-scale graphs. *Proc. VLDB Endow. 15*, 1572–1580. 105

Zhou, K., H. Zha, and L. Song (2013). Learning triggering kernels for multi-dimensional hawkes processes. In *International conference on machine learning*, pp. 1301–1309. 23

Zhou, L., Y. Yang, X. Ren, F. Wu, and Y. Zhuang (2018). Dynamic network embedding by modeling triadic closure process. In *Thirty-Second AAAI Conference on Artificial Intelligence*. 6

Zhu, L., D. Guo, J. Yin, G. Ver Steeg, and A. Galstyan (2016). Scalable temporal latent space inference for link prediction in dynamic social networks. *IEEE Transactions on Knowledge and Data Engineering 28*(10), 2765–2777. 6

Zuo, S., H. Jiang, Z. Li, T. Zhao, and H. Zha (2020). Transformer hawkes process. In *Proceedings of the 37th International Conference on Machine Learning*, pp. 11692–11702. 4, 26