TUM School of Computation, Information and Technology
Technische Universität München

Modeling Continuous-time Event Data with Neural Temporal Point Processes

Oleksandr Shchur

Vollständiger Abdruck der von der TUM School of Computation, Information and Technology der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Vorsitzender:
Prof. Dr. Nils Thuerey

Prüfende der Dissertation:
1. Prof. Dr. Stephan Günnemann
2. Prof. Dr. Scott Linderman,
   Stanford University

Die Dissertation wurde am 04.07.2022 bei der Technischen Universität München eingereicht und durch die TUM School of Computation, Information and Technology am 14.11.2022 angenommen.

Abstract

Event data lies at the core of many high-impact applications of machine learning. Hospital visits in electronic health records, earthquake catalogs in seismology, and spike trains in neuroscience — all can be represented as variable-length event sequences in continuous time. Temporal point processes (TPPs) provide a natural framework for modeling such data. However, conventional TPP models lack the ability to capture complex patterns present in real-world event data. Neural TPPs aim to address this limitation by combining neural networks with the fundamental ideas from point process literature. The two main themes of this thesis are (1) design of flexible, tractable and efficient neural TPP models, and (2) their applications to real-world problems. Our first contribution is the connection between TPPs and the field of neural density estimation. This allows us to develop the first neural TPP model, where likelihood computation, sampling, and prediction can all be done efficiently in closed form. Next, we propose TriTPP — a new class of expressive TPP models, where, unlike existing methods, all operations can be done in parallel. Fast parallel sampling opens new applications for TPP models. We show this by deriving a variational inference scheme for continuous-time discrete-state systems. Finally, we combine goodness-of-fit testing approaches with neural TPP models to create a simple and effective anomaly detection method for event sequences.

Zusammenfassung

Zeitlich aufgelöste Daten bilden einen wichtigen Bestandteil vieler Anwendungen des maschinellen Lernens. Seien es Krankenhausbesuche erfasst elektronischen Krankenakten, Erdbebenkataloge in der Seismologie oder Spike Trains in der Neurowissenschaft, all diese Daten lassen sich als Ereignisfolgen darstellen. Den natürlichen Rahmen zur Modellierung der zeitlichen Entwicklung von Ereignisfolgen bilden stochastische Prozesse oder genauer Temporale Punktprozesse (TPPs). Während konventionelle TPP-Modelle limitiert darin sind, komplexe Muster aus realen Ereignisdaten zu erfassen, können neuronale TPPs diese Einschränkung beheben, indem sie neuronale Netze mit den Methoden der Punktprozesstheorie kombinieren. Den Schwerpunkt dieser Arbeit bilden (1) der Entwurf flexibler, interpretierbarer und effizienter neuronaler TPP-Modelle und (2) die Anwendung dieser auf reale Probleme. Hierbei ist die gemeinsame Betrachtung von TPPs und dem Gebiet der neuronalen Dichteschätzung ein erster Beitrag. Das Resultat hieraus ermöglicht die Entwicklung des ersten neuronalen TPP-Modells, bei dem Likelihood-Berechnung, das Sampling und die Vorhersage effizient und in geschlossener Form durchgeführt werden können. Darauf aufbauend wird eine neue Klasse von flexiblen TPP-Modellen, TriTPP, eingeführt, welche im Gegensatz zu bestehenden Methoden erlaubt, alle Operationen parallel durchzuführen. Das schnelle und parallele Sampling eröffnet auf diese Weise neue Anwendungen für TPP-Modelle, da es Variationsverfahren für zeitkontinuierliche Systeme mit diskreten Zuständen erlaubt. Zuletzt wird gezeigt, wie die Kombination von Goodness-of-Fit-Tests und neuronalen TPP-Modellen eine einfache und effektive Methode zur Erkennung von Anomalien in Ereignisfolgen darstellt.

Acknowledgments

First of all, I would like to thank Prof. Stephan Günnemann. I feel so lucky and grateful for the opportunity to pursue a PhD under your guidance. Thank you for being such an incredible advisor, for showing me how to become a better researcher and teacher, and, especially, for our stimulating brainstorming sessions and discussions. I really appreciate your mentorship and support.

I am grateful to everyone at the DAML group at TUM for your readiness to help, the interesting conversations, and the great time we spent together both in the university and outside. Special thanks to Leon Hetzel for helping me translate the abstract of the dissertation to German.

I would like to thank my collaborators at TUM Aleksandar Bojchevski, Daniel Zügner, Marin Bilos, and Nicholas Gao. I learned so much from each of you, and working with you was both inspiring and lots of fun.

Thank you to all the incredible people that I was fortunate to work with during my internships. I am grateful to Caner Türkmen, Tim Januschowski and Jan Gasthaus at AWS, and Max Nickel and Matt Le at FAIR for their mentorship.

I want to also thank my parents for their support and encouragement, and for giving me an opportunity to study in Germany.

Thank you Vera for always being there for me and bringing so much joy to my life. I am grateful for your unwavering support throughout these years, your help with designing posters and presentations, and our inspiring discussions about research. There is no way I could complete this journey without you.

Contents

Part I

Introduction

1 Introduction

1.1 Machine learning for continuous-time event data

Financial transactions, online communication, neural spike trains, earthquakes — various human-made and natural phenomena can be represented as sequences of events in continuous time. Probabilistic models for such event data known as temporal point processes (TPP) can be used to make predictions, find patterns and better understand the respective real-world systems. The theory of TPPs was developed in the 20th century in the seminal works of Feller [57, 58], Cox [35, 38], Lewis [105, 106], Hawkes [79], and Ogata [133, 134]. Thanks to the ubiquity of event data, TPPs became widely adopted both in scientific fields like seismology [82, 135] and neuroscience [46, 68], as well as in industries such as finance [9, 80] and healthcare [3, 56].

Last decades saw an explosion in both scale and complexity of event data encountered in practical applications. New techniques in seismology enable collection of rich, diverse datasets with millions of earthquakes [126, 192]. Online services like Twitter and Facebook capture social interactions on an unprecedented scale, and hosting providers such as AWS generate petabytes of data each day [165]. Analyzing this data can unlock immense value. However, conventional TPP models, like Poisson or self-exciting processes, are unable to capture the complex patterns present in such data. Moreover, the event sequences are often accompanied by additional attributes (e.g., locations) that are relevant for prediction tasks, but incorporating them into conventional TPPs requires tedious feature engineering. Dealing with these issues requires developing new TPP models that have the flexibility to represent complex patterns and are scalable enough to handle large diverse datasets.

In recent years a new class of models known as *neural TPPs* emerged to address the above challenges [170]. Neural TPPs combine the fundamental ideas from the theory of point process with deep learning approaches. Deep learning methods are based on neural networks — expressive function approximators defined via composition of differentiable transformations [72]. Neural-network-based approaches significantly advanced the state of the art in computer vision [101, 186], natural language processing [23, 48], machine learning on graphs [212] and a number of other fields. In this thesis, we study the application of deep learning to continuous-time event data.

In the context of neural TPPs, the flexibility of neural networks allows us to learn different patterns of event occurrence automatically from the data, instead of specifying them manually, as in conventional models. For instance, in a self-exciting process, observed events can only increase the rate of arrival of future events — a rather limiting assumption that does not hold in many real-world event datasets (e.g., inhibitory neurons in the brain decrease the firing rate [46]). In contrast, a neural TPP model can automatically learn both inhibiting and excitatory effects of different event types in a purely data-driven way.

The seminal works by Du et al. [52] and Mei & Eisner [119] in 2016 were the first to show the new possibilities opened by combining TPPs with neural networks. These were followed by a number of papers that proposed new model architectures and parameter estimation algorithms for neural TPPs [95, 108, 181, 198–200]. However, there remain a number of open questions related to both design and application of deep-learning-based TPP models.

1.2 Contributions and outline

Neural TPP models should meet a number of requirements to be successfully applied to real-world tasks. Expressiveness, tractability, efficient training and inference for such models are often at odds with each other, and existing neural TPP architectures make suboptimal tradeoffs between these properties. In the first part of the thesis, we focus on these aspects of TPP model design, which we formulate as our first research question:

Research Question 1: *How can we define flexible neural TPP models that are at the same time tractable and efficient?*

We start by reviewing the basics of probabilistic modeling and deep learning, as well as provide a self-contained introduction to TPPs from a machine learning perspective in Chapter 2. In Chapter 3, we discuss the limitations of existing neural TPP models and introduce a new class of models that address these shortcomings. By drawing connections to the field of neural density estimation, we construct flexible neural TPPs, where both likelihood computation and sampling can be done analytically. This is a major improvement compared to existing approaches, none of which satisfy all these criteria simultaneously. Next, in Chapter 4, we take a different path and show how all TPPs can be viewed through the lens of triangular maps. Based on this insight, we propose TriTPP — a new flexible and efficient TPP parametrization based on compositions of invertible transformations. Modern TPP architectures usually utilize autoregressive neural networks (e.g., RNNs and transformers) and therefore are inherently sequential. In contrast, in TriTPP both sampling and training can be done in parallel, which leads to massively improved efficiency. Moreover, efficient sampling with *reparametrization* opens new applications for TPPs. This leads directly to our second research question:

Research Question 2: *How can we apply neural TPPs to solve real-world problems?*

In Chapter 5, we show how the reparametrization trick allows us to efficiently train TPP models with sampling-based losses. Such loss functions for TPPs are typically discontinuous, which makes optimizing them with gradient-based methods impossible. To address this challenge, we introduce a differentiable relaxation for losses involving variable-length event sequences. To show the utility of this approach, we develop a variational inference scheme for continuous-time discrete-state systems like Markov jump processes. Finally, in Chapter 6, we tackle anomaly detection for event sequences with TPPs. We demonstrate how the anomaly detection problem — for arbitrary data types,

Table 1.1: List of own publications that this thesis is based on. Code and datasets for the respective publications are available at `https://github.com/shchur/[repository]`.

Ch.	Ref.	Title	Conference	Repository
2–3	[170]	Neural Temporal Point Processes: A Review	IJCAI 2021	N/A
3	[167]	Intensity-free Learning of Temporal Point Processes	ICLR 2020	`/ifl-tpp/`
4–5	[168]	Fast and Flexible Temporal Point Processes with Triangular Maps	NeurIPS 2020	`/triangular-tpp/`
6	[169]	Detecting Anomalous Event Sequences with Temporal Point Processes	NeurIPS 2021	`/tpp-anomaly-detection/`

not just event sequences — can be approached using goodness-of-fit tests for generative models. We combine this framework with our neural TPP model from Chapter 3, which leads to a simple and effective method for anomaly detection.

1.3 Own publications

The content of Chapters 3 to 6 is mostly based on papers previously published at international peer-reviewed conferences. We list these papers in Table 1.1. We also provide the full list of publications that the author was involved in during the PhD studies below:

[1] Aleksandar Bojchevski, Oleksandr Shchur, Daniel Zügner, and Stephan Günnemann. Netgan: Generating graphs via random walks. In *International Conference on Machine Learning*, 2018.

[2] Federico Monti, Oleksandr Shchur, Aleksandar Bojchevski, Or Litany, Stephan Günnemann, and Michael M Bronstein. Dual-primal graph convolutional networks. *Graph Embedding and Mining Workshop, ECML-PKDD*, 2018.

[3] Oleksandr Shchur, Marin Biloš, and Stephan Günnemann. Intensity-free learning of temporal point processes. In *International Conference on Learning Representations*, 2020. (cited on pages 5, 43, 45, 49, 50, 75, 83, and 146)

[4] Oleksandr Shchur, Aleksandar Bojchevski, Mohamed Farghal, Stephan Günnemann, and Yusuf Saber. Anomaly detection in car-booking graphs. In *International Conference on Data Mining Workshops, ICDM*, 2018.

[5] Oleksandr Shchur, Nicholas Gao, Marin Biloš, and Stephan Günnemann. Fast and flexible temporal point processes with triangular maps. In *Advances in Neural Information Processing Systems*, 2020. (cited on pages 5, 55)

[6] Oleksandr Shchur and Stephan Günnemann. Overlapping community detection with graph neural networks. *Deep Learning on Graphs Worshop, KDD*, 2019.

[7] Oleksandr Shchur, Maximilian Mumme, Aleksandar Bojchevski, and Stephan Günnemann. Pitfalls of graph neural network evaluation. *Relational Representation Learning Workshop, NeurIPS*, 2018.

[8] Oleksandr Shchur, Ali Caner Türkmen, Tim Januschowski, Jan Gasthaus, and Stephan Günnemann. Detecting anomalous event sequences with temporal point processes. *Advances in Neural Information Processing Systems*, 2021. (cited on pages 5, 7)

[9] Oleksandr Shchur, Ali Caner Türkmen, Tim Januschowski, and Stephan Günnemann. Neural temporal point processes: A review. *International Joint Conference on Artificial Intelligence*, 2021. (cited on pages 3, 5, 7, 65, and 83)

2 Background

In this chapter we go over the mathematical foundations of TPPs. We introduce notation and important concepts that will allow us to discuss the contributions of the thesis. The overview of TPPs is primarily based on the lecture notes by Rasmussen [152], tutorial by Gomez-Rodriguez & Valera [71] and the review paper by Shchur et al. [170]. For a rigorous measure-theoretic treatment of TPPs, see Daley & Vere-Jones [41, 42].

We provide an overview of the notation and abbreviations used throughout the thesis in Appendices A and B respectively.

2.1 Generative probabilistic modelling

Probabilistic models are the foundation of modern machine learning [16, 127]. Generative probabilistic models are a class of approaches that define a stochastic procedure for generating data, usually in a way that matches the behavior of some real-world system. For instance, in seismology we might be interested in defining a model that generates realistic aftershock sequences to better understand or forecast earthquake sequences occurring in nature. Even if the system that we model is deterministic on the macroscopic level (e.g., earthquakes, climate, server logs), treating it as stochastic allows us to account for unobserved variables and imprecise measurements.

Typically, a generative model is governed by a set of *parameters* $\boldsymbol{\theta}$ that need to be estimated using observed data $\mathcal{D}_{\text{train}}$ produced by the system under study. A prominent example is the *maximum likelihood* principle, where the parameters of the generative model are optimized to maximize the (logarithm of the) probability of observed data [4]

$$\boldsymbol{\theta}_{\text{MLE}} = \arg\max_{\boldsymbol{\theta}} \ \log p_{\boldsymbol{\theta}}(\mathcal{D}_{\text{train}}). \tag{2.1}$$

Once the parameters have been estimated, the model can be used in a number of ways. A common application is *conditional generation* — given a partially observed data instance, a generative model can provide distribution over the missing data or predict the future. Examples of this task include generating possible sentence continuations [23], inpainting missing parts of an image [204], and earthquake forecasting [21]. Sometimes, learned parameters of the model can help us *understand* real-world systems. For instance, by fitting a generative model to neural spike trains, it is possible to recover the connectivity structure between the neurons [112]. Finally, by testing the *goodness of fit* of the model to the data, we can assess the validity of scientific hypotheses (e.g., "Does a certain stimulus elicit a response in this part of the brain?") [46, 68] or detect anomalies in the data [169].

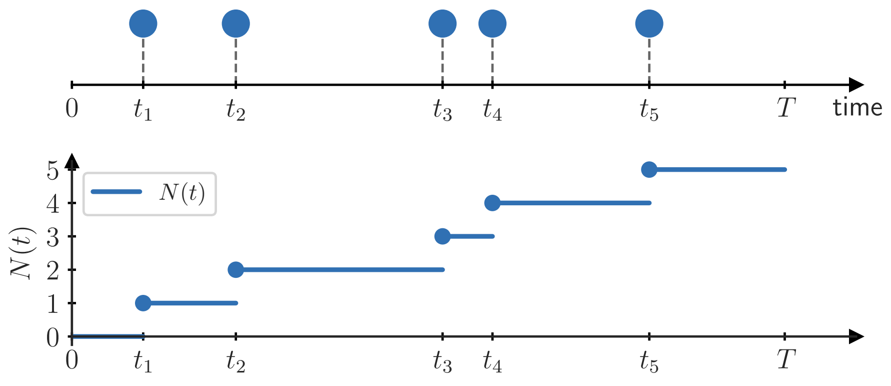


Figure 2.1: A realization of a TPP can be represented either as a sequence of arrival times $\mathcal{T} = (t_1, \dots, t_N)$ (above) or as a counting process $\{N(t) : t \in [0, T]\}$ (below).

2.2 Temporal point processes

2.2.1 Representation

In the most intuitive sense, a temporal point process (TPP) is a generative model for variable-length event sequences in continuous time. For example, suppose we would like to model the activity of a single user on a social network over some time window $[0, T]$. We can represent each post made by the user by its *arrival time* t_i and the activity over the entire time window as an *event sequence* $\mathcal{T} = (t_1, \dots, t_N)$. TPP allows us to define a probability distribution over such sequences, where both N, the number of events, as well as their arrival times t_i are random.[1] Note that it is also possible that no events are observed in the time window, that is $N = 0$. Sometimes it is convenient to instead work with the *inter-event times* $(\tau_1, \dots, \tau_{N+1})$, where we define $\tau_i := t_i - t_{i-1}$, assuming $t_0 = 0$ and $t_{N+1} = T$.

We can also view TPP as a type of stochastic process — i.e., a probability distribution over functions — known as a *counting process*. In this case, we represent a TPP realization by a function $N(t)$ that returns the number of events observed in the interval $[0, t]$. This is formally defined as

$$N(t) := \sum_{t_i \in \mathcal{T}} \mathbb{1}(t \geq t_i) \qquad \text{for } t \in [0, T]. \tag{2.2}$$

Here $\mathbb{1}(\cdot)$ is the indicator function that returns 1 if its argument is true and 0 otherwise. Figure 2.1 demonstrates the event sequence and counting process representations.

[1]We make the standard assumption that the point process is *simple* — that is, almost surely no two events happen at the exact same time. This means that the arrival times $t_1, \dots, t_N$ are strictly increasing, and therefore the inter-event times $\tau_1, \dots, \tau_{N+1}$ are strictly positive with probability 1.

2.2.2 TPP as an autoregressive model

Now we discuss how a TPP mathematically specifies the distribution over event sequences. One way to do this is to treat TPP as an *autoregressive model* and generate the arrival times one by one. More specifically, for each $i = 1, 2, 3, ...$ we need to specify $P_i(t_i|t_1, ..., t_{i-1})$ — the distribution of the next arrival time t_i conditioned on the past events $(t_1, \dots, t_{i-1})$. Since $P_i(t_i|t_1, ..., t_{i-1})$ is a continuous probability distribution, it can be defined by one of the following functions:

- probability density function (PDF)

$$p_i(t|t_1, ..., t_{i-1}) = \Pr(\text{event } t_i \in [t, t + dt] \mid t_1, ..., t_{i-1}) \tag{2.3}$$

  where dt denotes an infinitesimal change in t, formally defined as

$$:= \lim_{\Delta t \to 0} \frac{\Pr(\text{event } t_i \in [t, t + \Delta t] \mid t_1, ..., t_{i-1})}{\Delta t}. \tag{2.4}$$

- cumulative distribution function (CDF)

$$F_i(t|t_1, ..., t_{i-1}) = \Pr(\text{event } t_i \in [t_{i-1}, t] \mid t_1, ..., t_{i-1}) \tag{2.5}$$

- survival function (SF)

$$S_i(t|t_1, ..., t_{i-1}) = \Pr(\text{event } t_i \in [t, \infty) \mid t_1, ..., t_{i-1}) \tag{2.6}$$

- hazard function (HF)

$$\phi_i(t|t_1, ..., t_{i-1}) = \Pr(\text{event } t_i \in [t, t + dt] \mid t_1, ..., t_{i-1}, t_i \notin [t_{i-1}, t)) \tag{2.7}$$

- cumulative hazard function (CHF)

$$\Phi_i(t|t_1, ..., t_{i-1}) = \int_{t_{i-1}}^{t} \phi_i(t|t_1, ..., t_{i-1}) dt \tag{2.8}$$

PDF and CDF are well-known in machine learning literature, while SF, HF and CHF are commonly used in the field of survival analysis [94]. Most relevant to our discussion of TPPs are the PDF and HF. The key difference between these two is that the hazard function additionally conditions on the fact that the next event t_i has not occurred until time t.

Each of the above functions (Equations 2.3 to 2.8) uniquely specifies the conditional distribution $P_i(t_i|t_1, ..., t_{i-1})$ as well as the other four functions (see Appendix C for details). However, the choice of which function to parametrize can be very important in practice, as we discuss in detail in Chapter 3.

To summarize, we can define a TPP autoregressively by specifying a sequence of conditional distributions $\{P_1(t_1), P_2(t_2|t_1), P_3(t_3|t_1, t_2), ...\}$. We can do it, for example, by defining the conditional PDFs $\{p_1(t), p_2(t|t_1), p_3(t|t_1, t_2), ...\}$.

2.2.3 Conditional intensity function

An alternative to the autoregressive description of TPPs is based on the *conditional intensity function*. The conditional intensity, denoted as $\lambda^*(t)$, represents the instantaneous rate of arrival of new events at time t given the history of past events $\mathcal{H}(t) = \{t_j \in \mathcal{T} : t_j < t\}$. Formally, we define the conditional intensity as

$$\lambda^*(t) := \Pr(N(t + dt) - N(t) = 1 \mid \mathcal{H}(t)), \tag{2.9}$$

where the $*$ symbol reminds us of conditioning on the history $\mathcal{H}(t)$.

It is important to note that the history $\mathcal{H}(t)$ includes the information that no event happened in the interval $[t_{i-1}, t)$, assuming that t_{i-1} is the last event before time t. This fact allows us to see how the conditional intensity is connected to the autoregressive characterization of the TPP — we can define $\lambda^*(t)$ in a piecewise manner by stitching together the conditional hazard functions (Equation 2.7):

$$\lambda^*(t) = \begin{cases} \phi_1(t) & \text{if } 0 < t \le t_1, \\ \phi_2(t|t_1) & \text{if } t_1 < t \le t_2, \\ \vdots & \\ \phi_{N+1}(t|t_1, ..., t_N) & \text{if } t_N < t \le T. \end{cases} \tag{2.10}$$

Similarly, we can recover the conditional PDFs from the intensity as

$$p_i(t|t_1, ..., t_{i-1}) = \lambda^*(t) \exp\left(-\int_{t_{i-1}}^{t} \lambda^*(u) du\right). \tag{2.11}$$

In Appendix C we provide the formulas for obtaining other distribution functions.

For convenience, we assume that we always deal with *non-terminating* TPPs, where the next event will always happen at some point in the future. Mathematically, this corresponds to the condition $\int_t^\infty \lambda^*(u) du = \infty$ holding for any t and $\mathcal{H}(t)$. We additionally assume that the intensity is always strictly positive. Both these assumptions are just technicalities that simplify the theory when discussing simulation algorithms, but have little effect in practice — since we always work with TPPs on a bounded interval $[0, T]$, we can approximate a terminating TPP with zero intensity by pretending that the next event happens after time T with probability $1 - \varepsilon$, where ε is arbitrarily small.

2.2.4 Conventional TPP models

We will now discuss several instances of conventional (non-neural) TPP models. These are important for several reasons. First, these examples show how simple parametric intensity functions can produce different dynamics of event occurrence. More importantly, these examples provide a useful sanity check when developing new models — in later chapters we will use them to determine whether our neural TPPs are flexible enough to capture different patterns generated by the simpler conventional models.

Homogeneous Poisson process (HPP) is the TPP characterized by the constant intensity function

$$\lambda^*(t) = \mu, \tag{2.12}$$

where $\mu > 0$ is the rate parameter. From this definition of $\lambda^*(t)$ we can conclude that the rate of arrival of new events in the HPP does not depend on the history (known as the "memoryless property") and is constant over time. From these facts, we can derive several other properties of the HPP that are important both for theoretical analysis and design of algorithms:

(a) The number of events in any two disjoint intervals are independent.

(b) The number of events in any interval $[a, b]$ for $0 \leq a < b \leq T$ follows Poisson distribution with rate $\mu \cdot (b - a)$.

(c) Conditioned on N, the arrival times $(t_1, \dots, t_N)$ are independently and identically distributed (i.i.d.) according to the Uniform$([0, T])$ distribution.

(d) The inter-event times $(\tau_1, \tau_2, \dots)$ are i.i.d. random variables that follow the exponential distribution with rate μ.

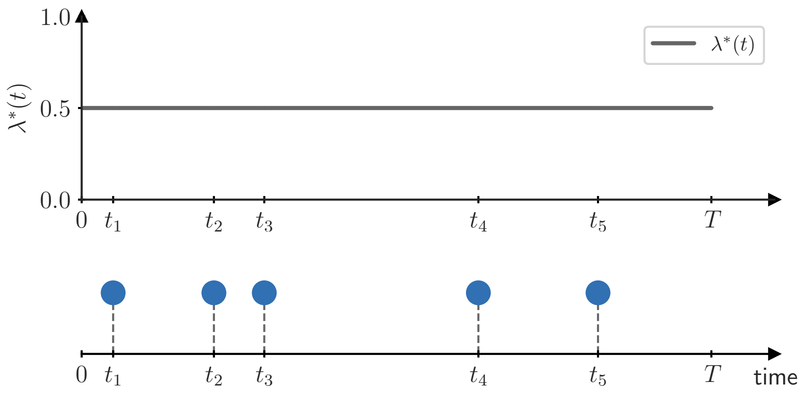


Figure 2.2: Homogeneous Poisson process with constant intensity $\lambda^*(t) = \mu$.

Standard Poisson process (SPP), also known as HPP with unit rate, defined by

$$\lambda^*(t) = 1, \tag{2.13}$$

is an important special case of the Poisson process. In many senses, the role of SPP in point process theory is similar to the role of the Uniform$([0, 1])$ distribution in univariate statistics. For instance, SPP is the highest entropy TPP on an interval $[0, T]$ [7]. More importantly, SPP can be used to generate samples from an arbitrary TPP, similar to how the Uniform$([0,1])$ distribution is used in inverse transform sampling for univariate random variables [49]. We discuss this aspect in more detail in Sections 2.2.5 and 2.2.7.

Inhomogeneous Poisson process (IPP) generalizes the homogeneous Poisson process by making the intensity vary over time:

$$\lambda^*(t) = \mu(t), \tag{2.14}$$

where $\mu(t)$ is a positive integrable function. Such formulation of the intensity preserves the memoryless property, like in the HPP, but is now able to capture periodic changes in the rate of arrival of new events. For example, IPP is a reasonable model for timestamps of calls arriving to a call center — the rate of activity may change throughout the day, but we can treat the calls as independent from each other. The inhomogeneous Poisson process also has a number of important properties:

(a) The number of events in any two disjoint intervals are independent.

(b) The number of events in any interval $[a, b]$ for $0 \le a < b \le T$ follows Poisson distribution with rate $\int_a^b \mu(u)du$.

(c) Conditioned on N, the arrival times $(t_1, \dots, t_N)$ are i.i.d. random variables with probability density function $f(t) = \mu(t)/\int_0^T \mu(u)du$.

Finally, it is worth noting that a TPP has the memoryless property if and only if it is a Poisson process.

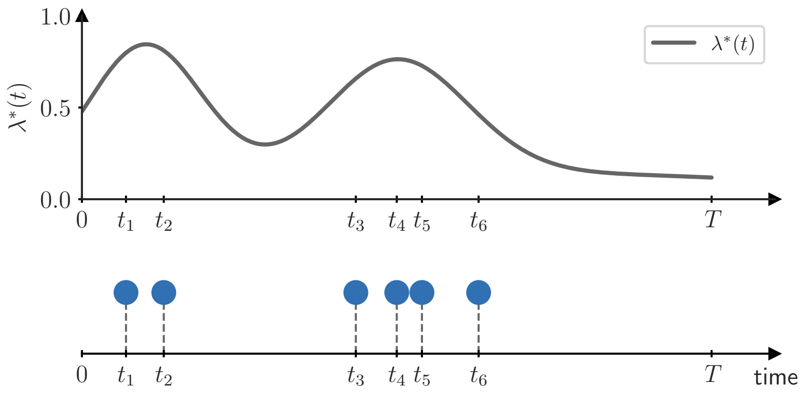


Figure 2.3: Inhomogeneous Poisson process has a history-independent intensity $\lambda^*(t) = \mu(t)$ and can model trends in the arrival rate of new events.

Renewal process (RP) [172] provides yet another generalization for the homogeneous Poisson process. Recall that in HPP the inter-event times $(\tau_1, \tau_2, \dots)$ are i.i.d. exponential random variables. RP replaces the exponential inter-event time distribution with another arbitrary distribution $P(\tau)$ supported on $[0, \infty)$. Suppose this inter-event time distribution is characterized by a hazard function $\varphi(\tau)$. Then, the conditional intensity function of the corresponding RP is

$$\lambda^*(t) = \varphi(t - t_{i-1}), \tag{2.15}$$

where t_{i-1} is the last observed event before time t (Figure 2.4).

One application where renewal processes naturally arise is reliability analysis. Suppose, we need to predict when a certain machine component that is prone to failures (e.g., lightbulb) needs to be replaced. Assuming that the time until failure for all lightbulbs follows the same distribution $P(\tau)$, and whenever one lightbulb breaks it is replaced by a new one, the sequence of failure times will follow the renewal process.

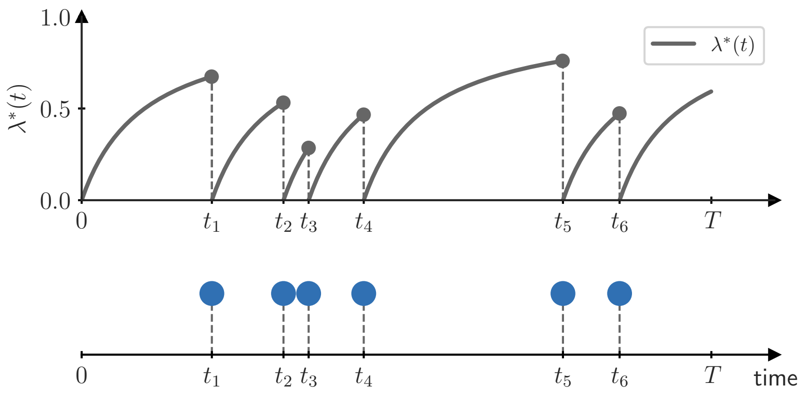


Figure 2.4: Renewal process with Gamma$(2, 1)$ inter-event time distribution.

Self-exciting process, also known as **Hawkes process (HP)** [79], has the property that its intensity increases whenever an event occurs. This leads to "bursty" event sequences, where multiple events often happen in quick succession. In the most general form, the conditional intensity of a Hawkes process is defined as

$$\lambda^*(t) = \mu(t) + \alpha \sum_{t_j \in \mathcal{H}(t)} \gamma(t - t_j). \tag{2.16}$$

The base rate $\mu(t)$ captures the seasonal change in event frequency, similar to IPP. The branching factor $\alpha \in (0, 1)$ represents the expected number of "offspring" events produced by each event. Finally, the triggering kernel $\gamma(\tau)$ (subject to $\int_0^\infty \gamma(\tau) d\tau = 1$) determines how the self-exciting effect decays over time.

One common choice is the exponential triggering kernel

$$\gamma(t - t_j) = \beta \exp(-\beta(t - t_j)) \tag{2.17}$$

with decay parameter $\beta > 0$ (Figure 2.5). Hawkes process with the exponential kernel is quite special, as it is the only self-exciting process that has the Markov property [132]. This means at any time t we can summarize the entire history $\mathcal{H}(t)$ with a single number that can later be reused when computing the intensity at a future time $t' > t$. The Markov property allows us to construct efficient algorithms for simulation [45] and parameter estimation [133] for Hawkes processes with the exponential kernel.

Other triggering kernels are also used in practice, but they lack the Markov property and thus require us to always keep track of all the past events $\mathcal{H}(t)$. For example, the power-law kernel

$$\gamma(t - t_j) = \frac{\eta c^\eta}{(t - t_j + c)^{1+\eta}} \tag{2.18}$$

is commonly used to model aftershocks in seismology [134] and information propagation on social media [158].

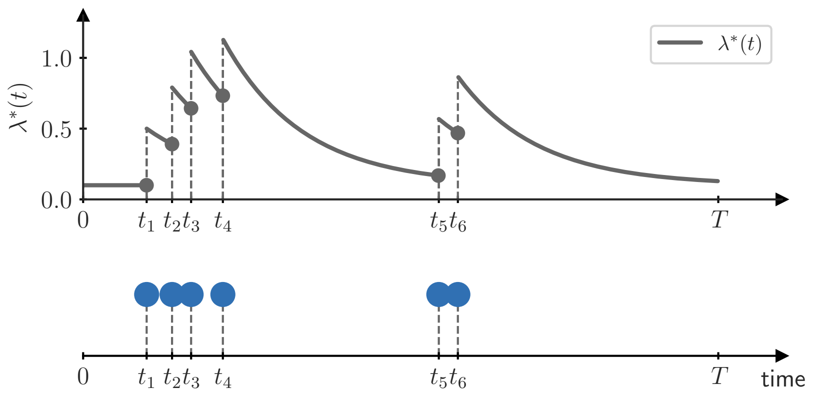


Figure 2.5: Self-exciting (Hawkes) process produces "bursty" event sequences. Here we use a constant base rate μ and exponential triggering kernel $\gamma(t - t_j) = \exp(-(t - t_j))$

Self-correcting process (SCP) [90] is in a sense the opposite of the self-exciting process — it allows observed events to inhibit future activity via the following intensity function

$$\lambda^*(t) = \exp\left(\mu t - \sum_{t_j \in \mathcal{H}(t)} \alpha\right) \tag{2.19}$$

with positive parameters μ, α. The intensity rises steadily over time, but decreases by a multiplicative factor of $e^{-\alpha}$ after each event, which produces sequences of evenly-spaced events (Figure 2.6). The SCP can be used to model events that rarely happen in quick succession, such as large-magnitude earthquakes.

Summary. It is worth highlighting that all TPPs that we discussed above can equivalently be specified using the conditional PDFs $p_i(t|t_1, \dots, t_{i-1})$ or the hazard functions $\phi_i(t|t_1, \dots, t_{i-1})$. Still, the description in terms of the conditional intensity $\lambda^*(t)$ happens to be more elegant and compact — we often do not have to worry about the indices i, and we can understand the properties of respective TPPs (such as global trends or

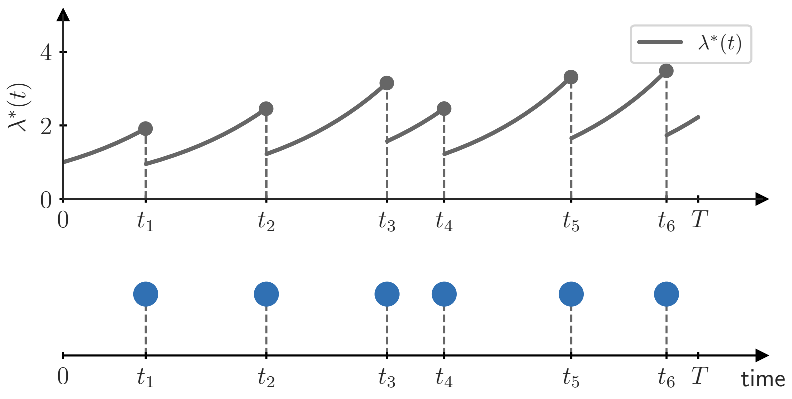


Figure 2.6: Self-correcting process produces evenly-spaced events.

burstiness) by inspecting the definition of $\lambda^*(t)$. In future chapters we will encounter other models, where intensity is not the most convenient representation anymore. However, we should always keep in mind that these different descriptions are interchangeable, and we can always move between them, if necessary.

2.2.5 Random time change theorem

We now present the random time change theorem — a central result in the theory of TPPs that lays the foundation for several contributions of this thesis. The random time change theorem shows us how we can connect an arbitrary TPP to the standard Poisson process using the conditional intensity $\lambda^*(t)$. This is similar to how in univariate statistics the CDF transform converts any continuous random variable on $\mathbb{R}$ into the uniform distribution [49].

Theorem 1 (Random time change theorem [22])**.** *Suppose $\mathcal{T} = (t_1, \dots, t_N)$ is a realization of a TPP with conditional intensity λ^* on an interval $[0, T]$. Define the transformation*

$$\Lambda^*(t) = \int_0^t \lambda^*(u) du. \tag{2.20}$$

Then the transformed sequence $\mathcal{Z} = (\Lambda^*(t_1), ..., \Lambda^*(t_N))$ is distributed according to the standard Poisson process on the interval $[0, \Lambda^*(T)]$.

The function $\Lambda^*(t)$, called the *integrated intensity function* or *compensator*, provides yet another way to characterize the TPP, in addition to the options listed in Sections 2.2.2 and 2.2.3. Therefore, we can express the compensator in terms of other functions characterizing the TPP, such as the cumulative hazard function (Equation 2.8):

$$\Lambda^*(t) = \Phi_i(t|t_1, \dots, t_{i-1}) + \sum_{j=1}^{i-1} \Phi_i(t_j|t_1, \dots, t_{j-1}), \tag{2.21}$$

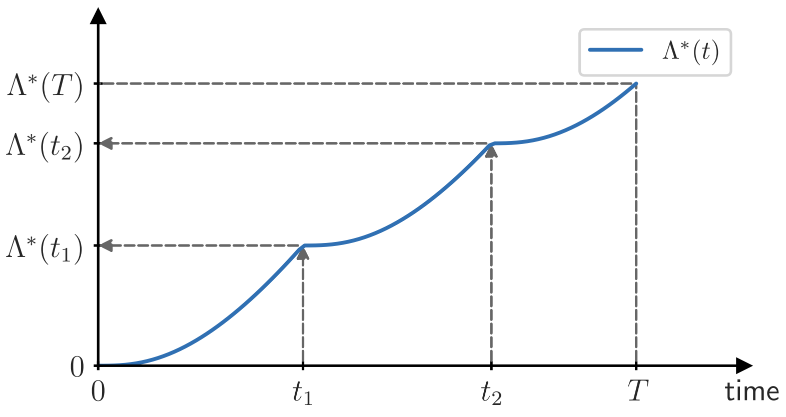


Figure 2.7: Compensator Λ^* transforms a realization $\mathcal{T} = (t_1, \dots, t_N)$ of a TPP on $[0, T]$ into a sample $\mathcal{Z} = (\Lambda^*(t_1), \dots, \Lambda^*(t_N))$ from the standard Poisson process on $[0, \Lambda^*(T)]$.

where t_{i-1} is the last event before time t.

Note that $\Lambda^* \colon [0, \infty) \to [0, \infty)$ is a strictly increasing invertible function since we defined it by integrating the strictly positive conditional intensity function of a non-terminating TPP. Invertability of the compensator will play an important role in our discussion of simulation algorithms for TPPs in Section 2.2.7. In Chapter 4 we will explore this property in more depth to define a new class of efficient and flexible neural TPP parametrizations. Moreover, the random time change theorem allows us to construct goodness-of-fit tests for arbitrary TPPs, which we will use to develop an anomaly detection framework in Chapter 6.

2.2.6 Parameter estimation

As we discussed in Section 2.1, applying a generative model usually requires estimating its parameters. Temporal point processes are no exception, and parameters of TPPs are typically learned using the maximum likelihood estimation (MLE) procedure.

Given an observed sequence $\mathcal{T} = (t_1, \dots, t_N)$, the *likelihood* of a TPP model with conditional intensity $\lambda^*(t)$ is computed as

$$p(\mathcal{T}) = \left(\prod_{i=1}^{N} \lambda^*(t_i)\right) \exp\left(-\int_0^T \lambda^*(t) dt\right). \tag{2.22}$$

Intuitively, this quantity can be interpreted as the probability that there are exactly N events before time T, one in each of the infinitesimal intervals $[t_i, t_i + dt]$. The function $p(\mathcal{T})$ is sometimes also called point process *density*. We will use $p(\mathcal{T})$ to refer both to the density function itself, as well as to the corresponding distribution over event sequences.

When estimating the parameters we instead work with the logarithm of the likelihood

$$\log p(\mathcal{T}) = \sum_{i=1}^{N} \log \lambda^*(t_i) - \int_0^T \lambda^*(t) dt. \tag{2.23}$$

In practice, the expression in Equation 2.23 is often divided by T, the length of the observed interval, to obtain values that are comparable across different sequences.

Recall that the conditional intensity function provides one of multiple equivalent ways to characterize a TPP. Similarly, the log-likelihood function can be equivalently expressed using other functions, e.g.,

$$\log p(\mathcal{T}) = \sum_{i=1}^{N} \log p_i(t_i|t_1, \dots, t_{i-1}) + \log S_{N+1}(T|t_1, \dots, t_N), \tag{2.24}$$

where $p_i(t|t_1, \dots, t_{i-1})$ are the conditional PDFs and $S_{N+1}(t|t_1, \dots, t_N)$ is the survival function of the event number $N + 1$.

The standard procedure for fitting a TPP model to a dataset consisting of i.i.d. event sequences $\mathcal{D}_{\text{train}} = \{\mathcal{T}^{(1)}, \dots, \mathcal{T}^{(M)}\}$ via MLE looks as follows. First, we pick some parametric form the conditional intensity $\lambda^*_{\boldsymbol{\theta}}(t)$ with parameters $\boldsymbol{\theta}$. For example, we may select the self-correcting process (Equation 2.19) with parameters $\boldsymbol{\theta} = \{\mu, \alpha\}$. Then, we find the parameters $\boldsymbol{\theta}_{\text{MLE}}$ that maximize the log-likelihood of the sequences in $\mathcal{D}_{\text{train}}$:

$$\boldsymbol{\theta}_{\text{MLE}} = \operatorname*{arg\,max}_{\boldsymbol{\theta} \in \Theta} \sum_{\mathcal{T} \in \mathcal{D}_{\text{train}}} \log p_{\boldsymbol{\theta}}(\mathcal{T}). \tag{2.25}$$

Here Θ encodes the potential constraints on the parameters (e.g., in self-correcting process μ and α must be positive), and the likelihood for each individual sequence is computed according to Equation 2.23. For some parametrizations of the TPP, it might be more convenient to implement the likelihood computation according to Equation 2.24 instead. Computing MLE analytically is only possible for simple models, such as HPP, so the optimization problem from Equation 2.25 is usually solved with numerical methods like gradient descent. Once we obtain the parameters by solving the above optimization problem, the learned model can be used for other prediction tasks.

While MLE is a simple and popular method for learning TPP models, there exist alternatives based on different objective functions and training procedures [108, 198, 202]. We will discuss these alternative approaches in detail in Chapter 5.

2.2.7 Simulation methods

Once we estimate the parameters of a TPP, we can use it to generate new event sequences. We will use the words *simulation* and *sampling* interchangeably to refer to the process of generating new event sequences from a TPP model. First, we consider generating event sequences "from scratch", that is, starting from time $t = 0$ and assuming no past events. At the end of this section, we will discuss conditional sampling, where we generate possible continuations of a partially-observed event sequence.

Different TPP simulation algorithms have different prerequisites, and therefore are suitable for different TPP models. We start by providing a simulation algorithm for the **homogeneous Poisson process** (Equation 2.12) that will serve as an important building block when generating samples from arbitrary TPPs. Recall that the inter-event times τ_i of HPP with rate μ are i.i.d. Exponential(μ) random variables (Property (d) of the HPP in Section 2.2.4). Algorithm 1 makes use of this property to simulate HPP on an aribtrary interval $[0, T]$.

Algorithm 1 Simulating a HPP on $[0, T]$ using inter-event times

```
Parameters: Interval length $T$, rate $\mu$.

1. Set $t = 0$, $i = 1$.
2. Repeat until $t > T$:
    (a) Sample the next inter-event time $\tau_i \sim \text{Exponential}(\mu)$.
    (b) Compute the next arrival time $t = t + \tau_i$.
    (c) If $t < T$, record the arrival time $t_i = t$ and increase the counter $i = i + 1$.

Output: Sequence of arrival times $\mathcal{T} = (t_1, \dots, t_N)$.
```

Algorithm 1 requires drawing samples from the exponential distribution. This can be done using the **inverse CDF transform** method for univariate random variables, as shown below.

Algorithm 2 Inverse CDF transform sampling for univariate random variables [49]

```
Parameters: Cumulative distribution function $F \colon \mathbb{R} \to [0, 1]$.

1. Sample $u \sim \text{Uniform}([0, 1])$
2. Solve $F(t) = u$ for $t$.

Output: Sample $t$ drawn from the distribution with CDF $F$.
```

The CDF $F(t) = (1 - \exp(-\mu t)) \cdot \mathbb{1}(t > 0)$ of the exponential distribution is invertible, which allows us to perform step 2 of Algorithm 2 analytically $t = F^{-1}(u) = -\frac{1}{\mu}\log(1-u)$.

Combining Algorithms 1 and 2 provides us with a concrete and easy-to-implement method for simulating the HPP. This, in turn, can be used to construct sampling methods for an arbitrary TPP with conditional intensity $\lambda^*(t)$. We will now discuss two such generic methods that are most commonly used in practice.

Inverse transform method for TPPs can be seen as the converse of the random time change theorem (Theorem 1). Recall that if $\mathcal{T} = (t_1, \dots, t_N)$ is a realization of a TPP with compensator Λ^* on $[0, T]$, then $\mathcal{Z} = (z_1, \dots, z_N) = (\Lambda^*(t_1), \dots, \Lambda^*(t_N))$ is a realization of the standard Poisson process on the interval $[0, \Lambda^*(T)]$. The main idea of this approach is to first simulate a sequence $\mathcal{Z} = (z_1, \dots, z_N)$ from the SPP and then obtain the actual arrival times as $\mathcal{T} = (t_1, \dots, t_N) = \left(\Lambda^{*-1}(z_1), \dots, \Lambda^{*-1}(z_N)\right)$. The main problem is that we do not know N, the number of events, in advance. However, since each t_i and the compensator $\Lambda^*(t_i)$ only depend on the previous arrival times $t_1, \dots, t_{i-1}$, we can generate the events one by one. The procedure is outlined in Algorithm 3.

Algorithm 3 Inverse transform sampling (via random time change theorem)

```
Parameters: Interval length $T$, compensator $\Lambda^*$.
  1. Set $t = 0$, $z = 0$ and $i = 1$.
  2. Repeat until $t > T$:
      (a) Sample the next SPP inter-event time $\nu_i \sim \text{Exponential}(1)$.
      (b) Compute the next SPP arrival time $z = z + \nu_i$.
      (c) Obtain the next observed arrival time as $t = \Lambda^{*-1}(z)$.
      (d) If $t < T$, record the arrival time $t_i = t$ and increase the counter $i = i + 1$.
Output: Sequence of arrival times $\mathcal{T} = (t_1, \dots, t_N)$.
```

Recall that since we consider non-terminating TPPs with strictly positive intensity, the compensator Λ^* is an invertible function (Section 2.2.5). Therefore, we use $\Lambda^{*-1}(z)$ to denote the unique solution to the equation $\Lambda^*(t) = z$.

Another way to understand the inverse transform method is by considering the autoregressive description of the TPP model. From this viewpoint, Algorithm 3 boils down to generating the arrival times t_i one by one from the respective conditional distributions $P_i(t_i | t_1, \dots, t_{i-1})$, for instance, using the inverse CDF transform.

Algorithm 4 Inverse transform sampling (via inverse CDF transform)

```
Parameters: Interval length $T$, arrival time CDFs $\{F_1(t), F_2(t|t_1), F_3(t|t_1, t_2), \dots\}$.
  1. Set $t = 0$ and $i = 1$.
  2. Repeat until $t > T$:
      (a) Sample $u \sim \text{Uniform}([0, 1])$.
      (b) Obtain the next observed arrival time as $t = F_i^{-1}(u | t_1, \dots, t_{i-1})$.
      (c) If $t < T$, record the arrival time $t_i = t$ and increase the counter $i = i + 1$.
Output: Sequence of arrival times $\mathcal{T} = (t_1, \dots, t_N)$.
```

The equivalence between Algorithms 3 and 4 can be proved using the relationships between CDF, CHF and the compensator (Appendix C). Finally, we note that the existence of the inverse conditional CDFs F_i^{-1} follows from the invertability of the compensator.

The inverse transform method provides a straightforward way to generate event sequences from an arbitrary TPP. The main requirement for applying this method in practice is the ability to efficiently sample from the conditional distribution $P_i(t_i|t_1, \dots, t_{i-1})$ (or, equivalently, invert the compensator). This, however, can be challenging for certain TPP parametrizations, such as Hawkes processes, where the compensator cannot be inverted analytically, and thus numerical root-finding methods are required.

Thinning algorithm [105, 133] provides an alternative that is computationally less efficient, but, unlike inverse transform, does not require inverting the compensator. The two prerequisites for applying the thinning algorithm are the ability to evaluate the intensity $\lambda^*(t)$ and a method to compute an upper bound on the intensity that fulfills

$$b^*(t) \geq \sup_{s \in [t, T]} \lambda^*(s). \tag{2.26}$$

Note that the upper bound also depends on past events $\mathcal{H}(t)$ that occured before time t. The main idea of the thinning algorithm is to first generate candidate events from a HPP that has higher intensity than the TPP of interest, and then to thin out some of the events to correct for oversampling. This is similar in spirit to the rejection sampling method from univariate statistics [49]. The procedure is described in Algorithm 5.

Algorithm 5 Ogata's modified thinning algorithm [133]

```
Parameters: Interval length $T$, conditional intensity $\lambda^*(t)$, upper bound $m(t)$.
  1. Set $t = 0$ and $i = 1$.
  2. Repeat until $t > T$:
      (a) Compute the upper bound $\mu_0 = b^*(t)$.
      (b) Generate an HPP inter-event time $\nu_i \sim \text{Exponential}(\mu_0)$.
      (c) Generate $u \sim \text{Uniform}([0, 1])$.
      (d) Compute candidate event as $t = t + \nu_i$.
      (e) If $t < T$ and $u < \frac{\lambda^*(t)}{\mu_0}$, accept the event $t_i = t$ and set $i = i + 1$.
Output: Sequence of arrival times $\mathcal{T} = (t_1, \dots, t_N)$.
```

The main disadvantage of the thinning method is that it may suffer from high rejection rates, and therefore be inefficient. This can happen if there is a lot of variation in the intensity, which results in low acceptance probability $\frac{\lambda^*(t)}{\mu_0}$. Moreover, obtaining an upper bound $b^*(t)$ may be challenging or even impossible for certain TPP parametrizations.

To summarize, both inverse transform and thinning permit exact simulation of TPPs, but have different prerequisites and involve different tradeoffs. When developing new TPP parametrizations, it is important to ensure that the requirements for at least one of the algorithms are met, so that the new model can be used for sampling. Otherwise an arbitrary neural TPP model may meet none of the prerequisites and therefore be of limited practical use.

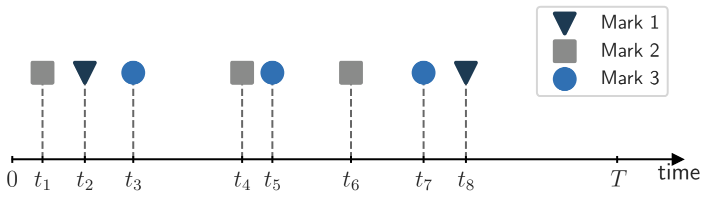


Figure 2.8: Marked TPP with $C = 3$ categorical marks corresponding to different event types.

Conditional generation. The simulation methods we discussed so far allow us to generate event sequences from scratch for the entire time interval $[0, T]$. However, this is not suitable for many practical applications, where the events up to some time $t_{\text{obs}} \in [0, T]$ are already observed. For instance, in earthquake forecasting the goal is usually to generate aftershock sequences in the forecast interval $(t_{\text{obs}}, T]$, conditioned on the previously observed earthquakes in $[0, t_{\text{obs}}]$.

Luckily, Algorithms 4 and 5 can both be easily adapted to the conditional setting. The thinning algorithm requires initializing $t = t_{\text{obs}}$ and conditioning on the observed events when computing $\lambda^*(t)$ and $b^*(t)$. For the inverse transform method, we need to change how we generate the first event. If N events were observed in $[0, t_{\text{obs}}]$, we need to account for the fact that event $N+1$ did not occur in the interval $[t_N, t_{\text{obs}}]$. This can be incorporated into Algorithm 4, e.g., by first computing the survival probability $\omega = 1 - F_{N+1}(t_{\text{obs}}|t_1, \dots, t_N)$ and then sampling the event as $F_{N+1}^{-1}(1 - \omega \cdot u_N | t_1, \dots, t_N)$.

2.2.8 Marked temporal point processes

So far we discussed event sequences where each event is represented only by its arrival time t_i. However, in many practical applications we have access to additional metadata associated with each event (e.g., magnitude and location of each earthquake). In this case, we denote each event by a tuple (t_i, m_i), where $m_i \in \mathcal{M}$ is a *mark* that contains the additional attributes. The definition of the mark space $\mathcal{M}$ depends on the type of the available metadata. For instance, we can model earthquakes with magnitude, longitude and latitude with $\mathcal{M} = (0, \infty) \times \mathbb{R}^2$. Another important case are categorical marks $\mathcal{M} = \{1, \dots, C\}$ that allow us to model events of C different types (Figure 2.8). This, for example, can be used to model the activity of C distinct users on a social network, each represented by their own mark.

Most of the previously discussed concepts can be straightforwardly extended to the marked case. We represent a marked TPP realization as $\mathcal{T} = \big((t_1, m_1), \dots, (t_N, m_N)\big)$, and the history now also includes the past marks $\mathcal{H}(t) = \{(t_j, m_j) : t_j < t\}$. A marked TPP can be defined autoregressively, i.e., by specifying the distribution of the next arrival time t_i and mark m_i conditioned on the history

$$p_i^*(t, m) := p_i\left(t, m | (t_1, m_1), \dots, (t_{i-1}, m_{i-1})\right) \qquad \text{for } i = 1, 2, 3, \dots. \tag{2.27}$$

Here we slightly abused the notation and for brevity used the $*$ symbol to denote conditioning on the past events.[2] We can decompose the conditional density as

$$p_i^*(t, m) = p_i^*(t) \cdot p_i^*(m|t_i = t). \tag{2.28}$$

Here $p_i^*(m|t_i = t)$ is either a probability density function or a probability mass function, depending on whether the marks are continuous or discrete. The conditional intensity for a marked TPP can be similarly expressed as

$$\lambda^*(t, m) = \lambda^*(t) \cdot p_i^*(m|t_i = t). \tag{2.29}$$

This means, we can model the distribution of the arrival times using $\lambda^*(t)$, analogous to how we did it for the unmarked case in Section 2.2.4, except that now the intensity $\lambda^*(t)$ may also depend on past marks.

In a general marked TPP, we usually condition the mark m on the arrival time t, as we just saw in Equations 2.28 and 2.29 (of course, both m and t are also conditioned on the history $\mathcal{H}(t)$, as indicated by $*$). However, in case of categorical marks, it may be more convenient to model the distribution of the arrival times separately for each mark. This can be done by directly specifying an individual conditional intensity function $\lambda_c^*(t)$ for each mark type $c \in \{1, \dots, C\}$

$$\lambda_c^*(t) := \lambda^*(t, m = c). \tag{2.30}$$

To make this idea more concrete, let us consider a specific example of the **multivariate Hawkes process**, where for each mark c we define the intensity as

$$\lambda_c^*(t) = \mu_c + \sum_{(t_j, m_j) \in \mathcal{H}(t)} A_{m_j, c} \cdot \gamma(t - t_j). \tag{2.31}$$

This definition is similar to the (unmarked) Hawkes process with exponential kernel that we discussed in Section 2.2.4, but with a few important differences. First, each mark now has its own base rate μ_c. More importantly, the $C \times C$ nonnegative influence matrix A allows us to model interactions between different event types: Every time an event of type l occurs, the intensity of mark c increases by $A_{l,c}$ and then fades according to the decay kernel $\gamma(\cdot)$.

Given the conditional intensity $\lambda_c^*(t)$ for each mark c, we can, for example, recover $\lambda^*(t)$, the rate of arrival of the next event of *any* type, by summing the intensities $\lambda^*(t) = \sum_{c=1}^{C} \lambda_c^*(t)$. Similarly, we can compute the probability that the next event that occurs at time t_i has specific type c as

$$p_i^*(m = c|t = t_i) = \frac{\lambda_c^*(t)}{\sum_{k=1}^{K} \lambda_k^*(t)}. \tag{2.32}$$

[2]Note that we overloaded the meaning of the $*$ symbol. In the conditional PDF $p_i^*(t, m)$ the $*$ denotes conditioning on past events $\big((t_1, m_1), \dots, (t_{i-1}, m_{i-1})\big)$, while in the conditional intensity $\lambda^*(t)$ the $*$ additionally includes the information that event t_i did not happen in the interval $[t_{i-1}, t]$.

Similar to the unmarked case, log-likelihood of a marked TPP can be expressed using either the conditional intensity or the conditional distributions as

$$\log p(\mathcal{T}) = \sum_{i=1}^{N} \log \lambda^*(t_i, m_i) - \int_0^T \lambda^*(u) du \tag{2.33}$$

$$= \sum_{i=1}^{N} \log p_i^*(t_i, m_i) + \log S_{N+1}^*(T) \tag{2.34}$$

and in case of categorical marks as

$$= \sum_{i=1}^{N} \log \lambda_{m_i}^*(t_i) - \sum_{c=1}^{C} \left(\int_0^T \lambda_c^*(u) du \right). \tag{2.35}$$

In the rest of this thesis we primarily focus on unmarked TPPs, unless explicitly stated otherwise. Still, many of the insights are transferable to the marked case thanks to the similarities we discussed above.

2.3 Deep learning for sequential data

The TPP models we have encountered so far are defined with hand-crafted parametric intensity functions that capture simple patterns of event occurrence (e.g., self-excitation). Neural TPPs take a different approach and parametrize the intensity using neural networks. This results in more flexible models that are able to learn various patterns from the data. We will now review some deep learning architectures that are commonly used when constructing neural TPP models. The contents of this section are primarily based on the textbook by Goodfellow et al. [72].

Multilayer perceptron (MLP), also known as **feedforward neural network** is the quintessential deep learning model. An MLP defines a function $g\colon \mathbb{R}^{D_{\text{in}}} \to \mathbb{R}^{D_{\text{out}}}$ operating on fixed-dimensional vectors $\boldsymbol{x} \in \mathbb{R}^{D_{\text{in}}}$ by stacking affine transformations of the form $\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}$ and element-wise activation functions $\sigma\colon \mathbb{R} \to \mathbb{R}$. For example, a two-layer MLP is defined as

$$g(\boldsymbol{x}) = \xi_2(\boldsymbol{W}_2 \xi_1(\boldsymbol{W}_1 \boldsymbol{x} + \boldsymbol{b}_1) + \boldsymbol{b}_2), \tag{2.36}$$

where $\boldsymbol{W}_1, \boldsymbol{W}_2, \boldsymbol{b}_1, \boldsymbol{b}_2$ are the learnable parameters (also known as weights) and ξ_1, ξ_2 are nonlinear activation functions, such as $\xi(x) = \tanh(x)$. MLPs embody one of the main principles of deep learning — they define a flexible function by composing simple transformations. Thanks to their simplicity and flexibility, MLPs are often used as building blocks in other deep learning architectures.

Recurrent neural network (RNN). When working with TPPs, we have to deal with variable-length event sequences. This requires neural network architectures that can operate with variable-length inputs and outputs. RNN is one of the simplest architectures with such property. We represent the input to the RNN as a sequence of vectors $(\boldsymbol{x}_1, \dots, \boldsymbol{x}_N)$. In a nutshell, an RNN processes the inputs $\boldsymbol{x}_i \in \mathbb{R}^{D_{\text{in}}}$ one by one and updates its hidden state $\boldsymbol{h}_i \in \mathbb{R}^H$. This way the hidden state $\boldsymbol{h}_{i+1}$ can extract relevant information from the previous inputs $(\boldsymbol{x}_1, \dots, \boldsymbol{x}_i)$.

More specifically, an RNN starts with the initial hidden state vector $\boldsymbol{h}_1$ and recursively compute the next state as

$$\boldsymbol{h}_{i+1} = \text{Update}(\boldsymbol{h}_i, \boldsymbol{x}_i). \tag{2.37}$$

Different implementations of the Update$(\cdot, \cdot)$ function correspond to different RNN architectures.

One of the simplest variants is the **Elman RNN** [55] with update function

$$\boldsymbol{h}_{i+1} = \tanh(\boldsymbol{W}_h \boldsymbol{h}_i + \boldsymbol{W}_x \boldsymbol{x}_i + \boldsymbol{b}), \tag{2.38}$$

where $\boldsymbol{W}_h, \boldsymbol{W}_x, \boldsymbol{b}$ are the learnable parameters. This RNN architecture has several known limitations, such as its poor ability to capture long-range dependencies between inputs, as well possible optimization problems due to vanishing and exploding gradients.

Various other RNN versions have been proposed to address these shortcomings, two most notable examples being **long-short term memory (LSTM)** [86] and **gated recurrent unit (GRU)** [31]. The main idea of these architectures is to have learnable gates that control the flow of information and therefore enable learning long-range dependencies. For example, the GRU update function is implemented as

$$\begin{aligned}
\boldsymbol{r}_{i+1} &= \sigma(\boldsymbol{W}_{hr}\boldsymbol{h}_i + \boldsymbol{W}_{xr}\boldsymbol{x}_i + \boldsymbol{b}_r) \\
\boldsymbol{z}_{i+1} &= \sigma(\boldsymbol{W}_{hz}\boldsymbol{h}_i + \boldsymbol{W}_{xz}\boldsymbol{x}_i + \boldsymbol{b}_z) \\
\hat{\boldsymbol{h}}_{i+1} &= \tanh(\boldsymbol{W}_{xh}\boldsymbol{x}_i + \boldsymbol{r}_{i+1} \odot (\boldsymbol{W}_{hh}\boldsymbol{h}_i + \boldsymbol{b}_h)) \\
\boldsymbol{h}_{i+1} &= (1 - \boldsymbol{z}_{i+1}) \odot \boldsymbol{h}_i + \boldsymbol{z}_{i+1} \odot \hat{\boldsymbol{h}}_{i+1}.
\end{aligned} \tag{2.39}$$

Here, the update gate $\boldsymbol{z}_{i+1} \in (0, 1)^H$, defined using the sigmoid function $\sigma(x) = 1/(1 + \exp(-x))$, allows the network to ignore certain inputs $\boldsymbol{x}_i$ and therefore better retain information from previous steps.

These architectural differences, however, do not play an important role in our subsequent discussion of neural TPPs. The main takeaway message of this section is that an RNN can encode a variable-length sequence of inputs $(\boldsymbol{x}_1, \dots, \boldsymbol{x}_i)$ into a fixed-dimensional state vector $\boldsymbol{h}_{i+1}$. We can treat the choice of the specific architecture (e.g., GRU vs. LSTM) as one of the hyperparameters of a neural TPP model. Similarly, we can replace RNN with other deep learning architectures for sequential data, such as **transformers** [185].

We have discussed all the important prerequisites and are ready to discuss how these deep learning architectures can be combined with the ideas from point process theory that we introduced in Section 2.2 to create flexible neural TPP models.

Part II

Neural temporal point process models

3 Intensity-free learning of temporal point processes

In this section, we begin our discussion of neural TPPs — generative models for event data based on neural networks. The main advantage of neural TPPs is their high expressiveness compared to traditional TPP models such as self-exciting or self-correcting processes (Section 2.2.4). Designing neural TPPs, however, usually involves trade-offs along the following dimensions:

- *Flexibility*: Can the model approximate any distribution?
- *Efficiency*: Can the likelihood function be evaluated in closed form?
- *Ease of use*: Can we easily sample new sequences and make predictions?

Existing methods [52, 119, 139] defined in terms of the conditional intensity function typically fall short in at least one of these regards.

Instead of modeling the intensity function, we suggest treating the problem of learning in temporal point processes as an instance of conditional density estimation. By using tools from neural density estimation [15, 155], we can develop methods that have all of the above properties. To summarize, our contributions are the following:

- We connect the fields of temporal point processes and neural density estimation. We show how normalizing flows can be used to define flexible and theoretically sound models for learning in temporal point processes.
- We propose a simple mixture model that performs on par with the state-of-the-art methods. Thanks to its simplicity, the model permits closed-form sampling and moment computation.
- We demonstrate how the proposed models can be used for prediction, conditional generation, sequence embedding, and training with missing data.

3.1 Background

Throughout this chapter we will use notation and terminology from Chapter 2 (see Appendices A and B for a recap). Recall that a TPP can be specified by a parametric conditional intensity function $\lambda^*_{\boldsymbol{\theta}}(t)$ that defines the rate of arrival of new events conditioned on the history (Equation 2.9). The chosen parametrization of $\lambda^*_{\boldsymbol{\theta}}(t)$ determines the flexibility of the TPP model, that is, what patterns of event occurrence it is able to capture (e.g., global trends; intensity increasing or decreasing after observed events). The intensity also plays an important role when learning the model parameters via maximum likelihood estimation (MLE). The goal of MLE is to find the parameters that maximize the log-likelihood of observed events $\mathcal{T} = (t_1, \dots, t_N)$ in the time interval $[0, T]$ that is computed as

$$\log p_{\boldsymbol{\theta}}(\mathcal{T}) = \sum_{i=1}^{N} \log \lambda^*_{\boldsymbol{\theta}}(t_i) - \int_0^T \lambda^*_{\boldsymbol{\theta}}(u) du. \tag{3.1}$$

The main challenge when picking a parametric form for $\lambda^*_{\boldsymbol{\theta}}(t)$ is the following tradeoff: For a "simple" intensity function, the integral $\int_0^T \lambda^*_{\boldsymbol{\theta}}(u)du$ has a closed form, which makes the log-likelihood easy to compute. However, such models typically have limited flexibility. A more sophisticated intensity parametrization can better capture the dynamics of the system but will often require approximating the integral numerically (e.g., with Monte Carlo), leading to inefficient optimization. To demonstrate these tradeoffs more concretely, we consider several existing neural TPP models.

Recurrent Marked Temporal Point Process (RMTPP) [52] was the first deep-learning-based TPP model. RMTPP consists of two components: First, an RNN is used to encode the history $\mathcal{H}(t_i) = \{t_1, \dots, t_{i-1}\}$ into a fixed-dimensional vector $\boldsymbol{h}_i$. Then, a decoder uses the history embedding $\boldsymbol{h}_i$ to parametrize the conditional intensity function. We focus on the decoder as it is most relevant to our discussion. Assuming that t_{i-1} is the last event that occurred before time t, the intensity is computed as

$$\lambda^*(t) = \exp(\boldsymbol{v}^T \boldsymbol{h}_i - w(t - t_{i-1}) + b), \tag{3.2}$$

where $\boldsymbol{v}, w, b$ are learnable parameters. The intensity of RMTPP can be integrated analytically, which leads to efficient MLE training. Moreover, the compensator $\Lambda^*(t)$ can be easily inverted, enabling inverse transform sampling (Algorithm 3). This, however, sacrifices flexibility, as Equation 3.2 can only represent a monotonically increasing or decreasing intensity between events.

Neural Hawkes Process [119] takes a different approach and instead encodes the history with a state $\boldsymbol{h}(t)$ that evolves in continuous time in addition to discrete updates after each observed event. The state $\boldsymbol{h}(t)$ then defines the intensity as

$$\lambda^*(t) = \text{softplus}\left(\boldsymbol{v}^T \boldsymbol{h}(t)\right). \tag{3.3}$$

Here, $\boldsymbol{v}$ is a learnable parameter and the softplus function $\log(1 + \exp(x))$ ensures positivity of the intensity. Such parametrization is more flexible than in case of RMTPP but, unfortunately, cannot be integrated analytically. This also means that we must resort to the potentially inefficient thinning algorithm (Algorithm 5) to sample new sequences with NHP.

Fully Neural Network Point Process (FullyNN) [139] is a recently proposed flexible, yet computationally tractable model for TPPs. The key idea of the FullyNN model is to instead parametrize the cumulative hazard function $\Phi_i(t|t_1, \dots, t_{i-1})$ with a monotonic neural network. The intensity is then defined as the derivative of the

	RMTPP	Neural Hawkes	Fully NN	Normalizing Flows	Mixture Distribution
Closed-form likelihood	✓	✗	✓	✓	✓
Flexible	✗	✓	✓	✓	✓
Closed-form $\mathbb{E}[\tau]$	✗	✗	✗	✗	✓
Closed-form sampling	✓	✗	✗	✗	✓

Table 3.1: Comparison of neural TPP models that encode history with an RNN.

cumulative hazard function, which allows us to efficiently compute the log-likelihood. Still, in its current state, the model has downsides: it does not define a valid TPP, sampling cannot be done exactly, and the expectation has no closed form.[1]

This work. We show that the drawbacks of the existing approaches can be remedied by looking at the problem of learning in TPPs from a different angle. Conventional TPP models (e.g., Poisson, Hawkes, self-correcting processes) are usually defined in terms of the conditional intensity function $\lambda^*(t)$, and existing neural TPPs follow the same strategy. We suggest to instead define neural TPPs autoregressively and directly work with the conditional PDFs $p_i(t|t_1, \dots, t_{i-1})$. Modeling probability distributions with neural networks is a well-researched topic, that, surprisingly, is not usually discussed in the context of TPPs. By adopting this alternative point of view, we are able to develop new theoretically sound and effective methods (Section 3.2), as well as better understand the existing approaches (Section 3.3).

3.2 Models

For convenience, in this section we will represent event sequences using inter-event times $(\tau_1, \dots, \tau_{N+1})$ instead of arrival times $(t_1, \dots, t_N)$, as we did before. All previous results and notation naturally translate to this alternative representation. We will again use $*$ as a shortcut for conditioning on past events $p_i^*(\tau) := p_i(\tau|\tau_1, \dots, \tau_{i-1})$. For example, the log-likelihood (Equation 3.1) can be expressed as

$$\log p_{\boldsymbol{\theta}}(\mathcal{T}) = \sum_{i=1}^{N} \log p_i^*(\tau_i) + \log S_{N+1}^*(\tau_{N+1}), \tag{3.4}$$

where $p_i^*(\tau)$ is the conditional PDF of the next inter-event time τ_i and $S_{N+1}^*(\tau)$ is the conditional survival function of τ_{N+1}. We always assume that the conditional PDF and SF are governed by the parameters $\boldsymbol{\theta}$ but hide them in our notation for compactness.

The main idea of our approach is to define the TPP autoregressively by modeling $p_i^*(\tau)$ instead of working with the conditional intensity function. First, we assume for simplicity that each inter-event time τ_i is independent of the history (that is, $p_i^*(\tau) = p(\tau)$ for all i). In Section 3.2.1, we show how state-of-the-art neural density estimation methods based

[1]A more detailed discussion of the FullyNN model follows in Section 6.4 and Appendix D.3.

on normalizing flows can be used to model $p(\tau)$. Then in Section 3.2.2, we propose a simple mixture model that can match the performance of the more sophisticated flow-based models, while also addressing some of their shortcomings. Finally, we discuss how to make $p_i^*(\tau)$ depend on the past events $(\tau_1, \dots, \tau_{i-1})$ in Section 3.2.3.

3.2.1 Modeling $p(\tau)$ with normalizing flows

The core idea of normalizing flows [155, 176] is to define a flexible probability distribution by transforming a simple one. Assume that y is distributed according to a PDF $q(u)$. Let $x = g(u)$ for some differentiable invertible transformation $g : \mathcal{U} \to \mathcal{X}$ (where $\mathcal{U}, \mathcal{X} \subseteq \mathbb{R}$).[2] We can obtain the PDF $p(x)$ of x using the change of variables formula as

$$p(x) = q(g^{-1}(x)) \left| \frac{\partial}{\partial x} g^{-1}(x) \right|. \tag{3.5}$$

By stacking multiple transformations $g_1, ..., g_M$, we obtain an expressive probability density function $p(x)$.

To draw a sample $x \sim p(x)$, we need to draw $u \sim q(u)$ and compute the *forward* transformation $x = (g_M \circ \cdots \circ g_1)(u)$. To compute the density at an arbitrary point x, it is necessary to evaluate the *inverse* transformation $u = (g_1^{-1} \circ \cdots \circ g_M^{-1})(x)$ and compute $q(u)$. Modern normalizing flows architectures parametrize the transformations using flexible functions $f_{\boldsymbol{\psi}}$, such as polynomials [91] or neural networks [88]. The flexibility of these functions comes at a cost — while the inverse $f_{\boldsymbol{\psi}}^{-1}$ exists, it typically does not have a closed form. That is, if we use such a function to define one direction of the transformation in a flow model, the other direction can only be approximated numerically using iterative root-finding methods [85]. In this work, we do not consider invertible normalizing flows based on dimension splitting, such as RealNVP [50], since they are not applicable to 1D data.

In the context of TPPs, our goal is to model the PDF $p(\tau)$ of the inter-event times. In order to learn the parameters of $p(\tau)$ using maximum likelihood, we need to be able to evaluate the density at any point τ. For this we need to define the inverse transformation $g^{-1} := (g_1^{-1} \circ \cdots \circ g_M^{-1})$. First, we set $u_M = g_M^{-1}(\tau) = \log \tau$ to convert a positive $\tau \in \mathbb{R}_+$ into $u_M \in \mathbb{R}$. Then, we stack multiple layers of parametric functions $f_{\boldsymbol{\psi}} : \mathbb{R} \to \mathbb{R}$ that can approximate any transformation. We consider two choices for $f_{\boldsymbol{\psi}}$:

- deep sigmoidal flow (DSF) [88]

$$f^{DSF}(x) = \sigma^{-1}\left(\sum_{k=1}^{K} w_k \sigma\left(\frac{x - \mu_k}{s_k} \right) \right) \tag{3.6}$$

- sum-of-squares (SOS) polynomial flow [91]

$$f^{SOS}(x) = a_0 + \sum_{k=1}^{K} \sum_{p=0}^{R} \sum_{q=0}^{R} \frac{a_{p,k} a_{q,k}}{p+q+1} x^{p+q+1} \tag{3.7}$$

[2]All definitions can be extended to $\mathbb{R}^D$ for $D > 1$. We consider the one-dimensional case since our goal is to model the distribution of inter-event times $\tau \in \mathbb{R}_+$.

where $\boldsymbol{a}, \boldsymbol{w}, \boldsymbol{s}, \boldsymbol{\mu}$ are the transformation parameters, K is the number of components, R is the polynomial degree, and $\sigma(x) = 1/(1 + e^{-x})$. We denote the two variants of the model based on f^{DSF} and f^{SOS} building blocks as **DSFlow** and **SOSFlow** respectively. Finally, after stacking multiple $g_m^{-1} = f_{\boldsymbol{\theta}_m}$, we apply a sigmoid transformation $g_1^{-1} = \sigma$ to convert u_2 into $u_1 \in (0, 1)$ that follows $q(u_1) = \text{Uniform}([0, 1])$.

For both models, we can evaluate the inverse transformations $(g_1^{-1} \circ \cdots \circ g_M^{-1})$, which means the model can be efficiently trained via maximum likelihood. The density $p(\tau)$ defined by either DSFlow or SOSFlow model is extremely flexible and can approximate any distribution (Section 3.2.5). Unfortunately, this is not sufficient for some use cases. Generating samples from both models requires evaluating the forward transformation $(g_M \circ \cdots \circ g_1)$ at a point $u_1 \sim q(u_1)$. This, however, cannot be done analytically since the functions f^{DSF} and f^{SOS} do not have a closed-form inverse. We can approximate the inverse with numerical root finding by solving the equation $(g_1^{-1} \circ \cdots \circ g_M^{-1})(\tau) - u_1 = 0$ for τ, but this approach is slow and inexact. As another example, we may be interested in predicting $\mathbb{E}_p[\tau]$, the expected time until the next event. Again, flow-based models are not optimal, since for them $\mathbb{E}_p[\tau]$ does not in general have a closed form.

This raises the question: Can we design a model for $p(\tau)$ that is as expressive as the flow-based models, but in which sampling and computing moments is easy and can be done in closed form?

3.2.2 Modeling $p(\tau)$ with mixture distributions

While mixture models are commonly used for clustering, they can also be used for density estimation. Mixtures work especially well in low dimensions [118], which is the case in TPPs, where we model the distribution of one-dimensional inter-event times τ. Since the inter-event times τ are positive, we choose to use a mixture of log-normal distributions to model $p(\tau)$. The PDF of a log-normal mixture is defined as

$$p(\tau|\boldsymbol{w}, \boldsymbol{\mu}, \boldsymbol{s}) = \sum_{k=1}^{K} w_k \frac{1}{\tau s_k \sqrt{2\pi}} \exp\left(-\frac{(\log \tau - \mu_k)^2}{2s_k^2}\right) \tag{3.8}$$

where $\boldsymbol{w}$ are the mixture weights, $\boldsymbol{\mu}$ are the mixture means, and $\boldsymbol{s}$ are the scale parameters. Because of its simplicity, the log-normal mixture model has a number of attractive properties.

Moments. Since each component k has a finite mean, the mean of the entire distribution can be computed as $\mathbb{E}_p[\tau] = \sum_k w_k \exp(\mu_k + s_k^2/2)$, i.e., a weighted average of component means. Higher moments can be computed based on the moments of each component [64].

Sampling. While flow-based models from Section 3.2.1 require iterative root-finding algorithms to generate samples, sampling from the log-normal mixture can be done in closed form:

$$\boldsymbol{z} \sim \text{Categorical}(\boldsymbol{w}) \qquad \varepsilon \sim \text{Normal}(0, 1) \qquad \tau = \exp(\boldsymbol{s}^T \boldsymbol{z} \cdot \varepsilon + \boldsymbol{\mu}^T \boldsymbol{z})$$

where $\boldsymbol{z}$ is a one-hot vector of size K.

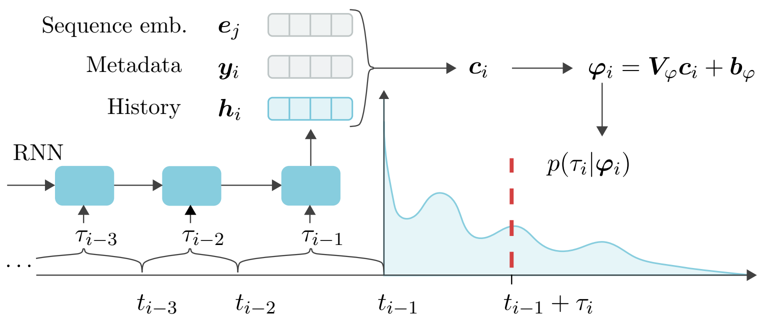


Figure 3.1: Model architecture. Parameters of $p_i^*(\tau|\varphi_i)$ are generated from the context vector c_i that consists of history embedding and additional attributes.

In some applications, such as reinforcement learning [183], we might be interested in computing gradients of the samples w.r.t. the model parameters. The samples τ drawn using the procedure above are differentiable with respect to the means μ and scales s. We can obtain gradients w.r.t. all the model parameters by using the Gumbel-softmax trick [92] when sampling z. For this we replace sampling $z \sim \text{Categorical}(w)$ with the following procedure

$$z = \text{softmax}\left(\frac{\log w + o}{\zeta}\right),$$

where each o_k is sampled i.i.d. from the standard Gumbel distribution and $\zeta > 0$ is the temperature parameter. As ζ approaches zero, samples from the Gumbel-softmax distribution approach exact samples from the Categorical(w) distribution.

Such reparametrization gradients have lower variance and are easier to implement than the score function estimators used in other works [121]. Other flexible models (such as multi-layer flow models from Section 3.2.1) do not permit sampling through reparametrization, and thus are not well-suited for the above-mentioned scenario. In Section 3.4.4, we show how reparametrization sampling can also be used to train with missing data by performing imputation on the fly. In Chapter 5 we provide a detailed discussion of reparametrization sampling for TPPs and the correspoding sampling-based losses.

3.2.3 Incorporating the conditional information

History. The key feature of TPPs is that the time $\tau_i = (t_i - t_{i-1})$ until the next event may be influenced by all the events that happened before. A standard way of capturing this dependency is to process the event history $\mathcal{H}(t_i)$ with a recurrent neural network (RNN) and embed it into a fixed-dimensional vector $h_i \in \mathbb{R}^H$. We accomplish this by following the procedure described by Du et al. [52].

The RNN starts with the initial hidden state h_1. Then, after each observed event τ_i, the new hidden state h_{i+1} is computed using the previous state h_i and the RNN input $x_i = [\log \tau_i]$ according to the RNN update equation

$$h_{i+1} = \text{Update}(h_i, x_i). \tag{3.9}$$

More specifically, we use the Elman RNN architecture (Equation 2.38) [55] in our experiments. The hidden state $\boldsymbol{h}_i$ serves as the history embedding that summarizes observed events $(\tau_1, \dots, \tau_{i-1})$.

Conditioning on additional features. The distribution of the time until the next event might depend on factors other than the history. For instance, distribution of arrival times of customers in a restaurant depends on the day of the week. As another example, if we are modeling user behavior in an online system, we can obtain a different distribution $p_i^*(\tau)$ for each user by conditioning on their metadata. We denote such side information as a vector $\boldsymbol{y}_i$. Such information is different from marks [152], since (a) the metadata may be shared for the entire sequence and (b) $\boldsymbol{y}_i$ only influences the distribution $p_i^*(\tau|\boldsymbol{y}_i)$, not the objective function.

In some scenarios, we might be interested in learning from multiple event sequences. In such case, we can assign each sequence $\mathcal{T}^{(j)}$ a learnable **sequence embedding** vector $\boldsymbol{e}_j$. By optimizing $\boldsymbol{e}_j$, the model can learn to distinguish between sequences that come from different distributions. The learned embeddings can then be used for visualization, clustering or other downstream tasks.

Obtaining the parameters. We model the conditional dependence of the distribution $p_i^*(\tau)$ on all of the above factors in the following way. The history embedding $\boldsymbol{h}_i$, metadata $\boldsymbol{y}_i$ and sequence embedding $\boldsymbol{e}_j$ are concatenated into a context vector $\boldsymbol{c}_i = [\boldsymbol{h}_i || \boldsymbol{y}_i || \boldsymbol{e}_j]$. Then, we obtain the parameters of the distribution $p_i^*(\tau)$ as an affine function of $\boldsymbol{c}_i$. For example, for the log-normal mixture model we have

$$\boldsymbol{w}_i = \operatorname{softmax}(\boldsymbol{V_w} \boldsymbol{c}_i + \boldsymbol{b_w}) \qquad \boldsymbol{s}_i = \exp(\boldsymbol{V_s} \boldsymbol{c}_i + \boldsymbol{b_s}) \qquad \boldsymbol{\mu}_i = \boldsymbol{V_\mu} \boldsymbol{c}_i + \boldsymbol{b_\mu} \tag{3.10}$$

where the softmax$(\cdot)$ and exp$(\cdot)$ transformations are applied to enforce the constraints on the distribution parameters, and $\{\boldsymbol{V_w}, \boldsymbol{V_s}, \boldsymbol{V_\mu}, \boldsymbol{b_w}, \boldsymbol{b_s}, \boldsymbol{b_\mu}\}$ are learnable parameters. Such model resembles the mixture density network architecture [15]. The whole process is illustrated in Figure 3.1. We obtain the parameters of the flow-based models in a similar way (see Appendix D.4).

3.2.4 Marked TPP

The flow-based and mixture-based models introduced above can be naturally extended to marked TPPs (Section 2.2.8). In a marked TPP, each event is represented by a mark $m_i \in \mathcal{M}$ in addition to the inter-event time τ_i. Different mark types $\mathcal{M}$ require different modifications to the model architecture (i.e., the history encoder, parametrization of the conditional distribution and the loss).

Categorical marks correspond to $\mathcal{M} = \{1, \dots, C\}$ and can be used to model events of C different categories. To feed such marks as input to the RNN history encoder, we can use an embedding layer [52]. That is, we define a matrix $\boldsymbol{V} \in \mathbb{R}^{C \times D_{\text{emb}}}$, where each row $\boldsymbol{V}_{c,:}$ corresponds to a learnable embedding of the respective mark. We obtain the RNN input for event (τ_i, m_i) as $\boldsymbol{x}_i = [\log \tau_i || \boldsymbol{V}_{m_i,:}]$. This choice allows the history embedding $\boldsymbol{h}_i$ to incorporate the information about marks of past events $\mathcal{H}(t_i)$.

There exist two ways to specify the conditional distribution of the next event. The simpler option is to make τ_i and m_i **conditionally independent** given the history.

This means, we do not permit any interactions between τ_i and m_i and model their distributions independently $p_i^*(\tau, m) = p_i^*(\tau) \cdot p_i^*(m)$ using the context vector $\boldsymbol{c}_i$. We can define the probability mass function (PMF) $p_i^*(m)$ of the next mark m_i with a categorical distribution, where the logits are computed from $\boldsymbol{c}_i$ similar to Equation 3.10. We model the distribution $p_i^*(\tau)$ of the next inter-event time τ_i exactly like we did in Section 3.2.1. We can implement the log-likelihood computation of the conditionally independent model as

$$\log p_{\boldsymbol{\theta}}(\mathcal{T}) = \sum_{i=1}^{N} \big(\log p_i^*(\tau_i) + \log p_i^*(m_i) \big) + \log S_{N+1}^*(\tau_{N+1}). \tag{3.11}$$

When generating new events from this model, we sample τ_i and m_i independently from the respective distributions.

The main limitation of the conditionally independent model is that the probability of observing an event of a specific type does not change over time. We can overcome this limitation by defining a model with a **separate distribution per mark**. For this we need to define a separate PDF $p_i^*(\tau|m = c)$ for each event type $c \in \{1, \dots, C\}$ — for example, by obtaining C different sets of log-normal mixture parameters using different model weights in Equation 3.10. The log-likelihood computation for this model can be implemented as

$$\log p_{\boldsymbol{\theta}}(\mathcal{T}) = \sum_{i=1}^{N} \left(\log p_i^*(\tau_i|m = m_i) + \sum_{\substack{c=1 \\ c \neq m_i}}^{C} \log S_i^*(\tau_i|m = c) \right) + \log S_{N+1}^*(\tau_{N+1}). \tag{3.12}$$

We highlight that the various expressions for marked TPP log-likelihood (Equations 3.11, 3.12, as well as 2.33–2.35) correspond to the exact same objective function. However, choosing different formulations for different models allows us to efficiently implement the log-likelihood computation on a computer. We can generate new events from the "separate distribution" model as follows:

1. For each mark $c \in \{1, \dots, C\}$, sample a candidate time τ_{ic} from the respective conditional distribution with PDF $p_i^*(\tau|m = c)$.

2. Set the next observed event (τ_i, m_i) as the "earliest" of the candidates τ_{ic}

$$\tau_i = \min_{c \in \{1,\dots,C\}} \tau_{ic} \qquad\qquad m_i = \operatorname*{arg\,min}_{c \in \{1,\dots,C\}} \tau_{ic}.$$

Continuous marks correspond to $\mathcal{M} = \mathbb{R}^C$ and can be used, for example, to model locations of events. We can concatenate the marks to the RNN input as $\boldsymbol{x}_i = [\log \tau_i || m_i]$ or manually specify features based on m_i using domain knowledge. We can again use the conditionally independent approach to model $p_i^*(\tau)$ and $p_i^*(m)$ separately, like we did in Equation 3.11. A more flexible alternative is to model the joint density $p_i^*(\tau, m)$

of the next inter-event time and the mark with a normalizing flow or a mixture model. In this case, the log-likelihood can be computed as

$$\log p_{\boldsymbol{\theta}}(\mathcal{T}) = \sum_{i=1}^{N} \log p_i^*(\tau_i, m_i) + S_{N+1}^*(\tau_{N+1}). \tag{3.13}$$

3.2.5 Discussion

Universal approximation. The SOSFlow and DSFlow models can approximate any probability density on $\mathbb{R}$ arbitrarily well [91, Theorem 3], [88, Theorem 4]. It turns out, a mixture model has the same universal approximation (UA) property.

Theorem 1 *[44, Theorem 33.2]. Let $p(x)$ be a continuous density on $\mathbb{R}$. If $q(x)$ is any density on $\mathbb{R}$ and is also continuous, then, given $\varepsilon > 0$ and a compact set $\mathcal{S} \subset \mathbb{R}$, there exist number of components $K \in \mathbb{N}$, mixture coefficients $\boldsymbol{w} \in \Delta^{K-1}$, locations $\boldsymbol{\mu} \in \mathbb{R}^K$, and scales $\boldsymbol{s} \in \mathbb{R}_+^K$ such that for the mixture distribution $\hat{p}(x) = \sum_{k=1}^K w_k \frac{1}{s_k} q(\frac{x-\mu_k}{s_k})$ it holds $\sup_{x\in\mathcal{S}} |p(x) - \hat{p}(x)| < \varepsilon$.*

This results shows that, in principle, the mixture distribution is as expressive as the flow-based models. Since we are modeling the conditional density, we additionally need to assume for all of the above models that the RNN can encode all the relevant information into the history embedding $\boldsymbol{h}_i$. This can be accomplished by invoking the universal approximation theorems for RNNs [163, 171].

Note that this result, like other UA theorems of this kind [39, 43], does not provide any practical guarantees on the obtained approximation quality, and does not say how to learn the model parameters. Still, UA intuitively seems like a desirable property of a distribution. This intuition is supported by experimental results. In Section 3.4.1, we show that models with the UA property consistently outperform the less flexible ones.

Interestingly, Theorem 1 does not make any assumptions about the form of the base density $q(x)$. This means we could as well use a mixture of distribution other than log-normal, such as Weibull. However, many other popular distributions on $\mathbb{R}_+$ have drawbacks: log-logistic does not always have defined moments and gamma distribution does not permit straightforward sampling with reparametrization.

Intensity function. For both flow-based and mixture models, the conditional survival function $S_i^*(\tau)$ and PDF $p_i^*(\tau)$ are readily available. This means we can easily compute the respective intensity functions (see Appendix C). However, we should still ask whether we lose anything by modeling $p_i^*(\tau)$ instead of $\lambda^*(t)$. The main arguments in favor of modeling the intensity function in traditional models (e.g. self-exciting process) are that it is *intuitive*, *easy to specify* and *reusable* [182].

"Intensity function is *intuitive*, while the conditional density is not." — While it is true that in simple models (e.g., Hawkes processes) the dependence of $\lambda^*(t)$ on the history is intuitive and interpretable, modern RNN-based intensity functions (as in [52, 119, 139]) cannot be easily understood by humans. In this sense, our proposed models are as intuitive and interpretable as other existing intensity-based neural network models.

"$\lambda^*(t)$ is *easy to specify*, since it only has to be positive. On the other hand, $p_i^*(\tau)$ must integrate to one." — As we saw, by using either normalizing flows or a mixture distribution, we automatically enforce that the PDF integrates to one, without sacrificing the flexibility of our model.

"*Reusability:* If we merge two independent point processes with intensities $\lambda^*_{(1)}(t)$ and $\lambda^*_{(2)}(t)$, the merged process has intensity $\lambda^*(t) = \lambda^*_{(1)}(t) + \lambda^*_{(2)}(t)$." — An equivalent result exists for the SFs $S^*_{(1)}(\tau)$ and $S^*_{(2)}(\tau)$ of two independent processes. The SF of the merged process is obtained as $S^*(\tau) = S^*_{(1)}(\tau)S^*_{(2)}(\tau)$, which can be derived from the connection between the conditional intensity and the cumulative hazard function (Appendix C).

As we just showed, modeling $p^*_i(\tau)$ instead of $\lambda^*(t)$ does not impose any limitation on our approach. Moreover, a mixture distribution is flexible, easy to sample from and has well-defined moments, which favorably compares it to other intensity-based deep learning models.

3.3 Related work

Neural temporal point processes. Conventional TPP models (e.g. self-exciting [79] or self-correcting [90] processes) often provide poor fit to real-world dataset due to their low flexibility. Multiple recent works address this issue by proposing more flexible neural-network-based point process models. These neural models are usually defined in terms of the conditional intensity function. For example, Mei & Eisner [119] propose a novel RNN architecture that can model sophisticated intensity functions. This flexibility comes at the cost of inability to evaluate the likelihood in closed form, and thus requiring Monte Carlo integration.

Du et al. [52] suggest using an RNN to encode the event history into a vector $\boldsymbol{h}_i$. The history embedding $\boldsymbol{h}_i$ is then used to define the conditional intensity, for example, using the *constant intensity* model $\lambda^*(t_i) = \exp(\boldsymbol{v}^T\boldsymbol{h}_i + b)$ [89, 108] or the more flexible *exponential intensity* model $\lambda^*(t_i) = \exp(w(t_i - t_{i-1}) + \boldsymbol{v}^T\boldsymbol{h}_i + b)$ [52, 183]. By considering the conditional inter-event time distribution $p^*_i(\tau)$ of the two models, we can better understand their properties. Constant intensity corresponds to the exponential distribution, and exponential intensity corresponds to the Gompertz distribution (see Appendix D.2). Clearly, these unimodal distributions cannot match the flexibility of a mixture model (as can be seen in Figure D.1).

Omi et al. [139] introduce a flexible fully neural network (FullyNN) intensity model, where they model the cumulative hazard function $\Phi^*_i(\tau)$ with a neural net. The function Φ^*_i converts τ into an exponentially distributed random variable with unit rate [152], similarly to how normalizing flows model $p^*_i(\tau)$ by converting τ into a random variable with a simple distribution. However, due to a suboptimal choice of the network architecture, the FullyNN model assigns non-zero probability to *negative* inter-event times (see Appendix D.3). In contrast, SOSFlow and DSFlow always define a valid PDF on $\mathbb{R}_+$. Moreover, similar to other flow-based models, sampling from the FullyNN model requires iterative root finding.

Several works used mixtures of kernels to parametrize the conditional intensity function [138, 177, 178]. Such models can only capture self-exciting influence from past events. Moreover, these models do not permit computing expectation and drawing samples in closed form. Recently, Biloš et al. [26] and Türkmen et al. [181] proposed neural models for learning marked TPPs. These models focus on event type prediction and share the limitations of other neural intensity-based approaches. Other recent works consider alternatives to the maximum likelihood objective for training TPPs. Examples include noise-contrastive estimation [77], Wasserstein distance [199, 201, 202], and reinforcement learning [108, 183]. This line of research is orthogonal to our contribution, and the models proposed in our work can be combined with the above-mentioned training procedures.

Neural density estimation. There exist two popular paradigms for learning flexible probability distributions using neural networks: In mixture density networks [15], a neural net directly produces the distribution parameters; in normalizing flows [155, 176], we obtain a complex distribution by transforming a simple one. Both mixture models [54, 74, 164] and normalizing flows [140, 216] have been applied for modeling sequential data. However, surprisingly, none of the existing works make the connection and consider these approaches in the context of TPPs.

3.4 Experiments

We evaluate the proposed models on the established task of event time prediction (with and without marks) in Sections 3.4.1 and 3.4.2. In the remaining experiments, we show how the log-normal mixture model can be used for incorporating extra conditional information, training with missing data and learning sequence embeddings. We use 6 **real-world datasets** containing event data from various domains: Wikipedia (article edits), MOOC (user interaction with online course system), Reddit (posts in social media) [102], Stack Overflow (badges received by users), LastFM (music playback) [52], and Yelp (check-ins to restaurants). We also generate 5 **synthetic datasets** (Poisson, Renewal, Self-correcting, Hawkes1, Hawkes2), as described in [139]. Detailed descriptions and summary statistics of all the datasets are provided in Appendix D.5.

3.4.1 Event time prediction using history

Setup. We consider two normalizing flow models, **SOSFlow** and **DSFlow** (Equations 3.6 and 3.7), as well a log-normal mixture model (Equation 3.8), denoted as **LogNormMix**. As baselines, we consider **RMTPP** (i.e. Gompertz distribution / exponential intensity from [52]) and **FullyNN** model from Omi et al. [139]. Additionally, we use a single log-normal distribution (denoted **LogNormal**) to highlight the benefits of the mixture model. For all models, an RNN encodes the history into a vector $\boldsymbol{h}_i$. The parameters of $p_i^*(\tau)$ are then obtained using $\boldsymbol{h}_i$ (Equation 3.10). We exclude the NeuralHawkes model from our comparison, since it is known to be inferior to RMTPP in time prediction [119], and, unlike other models, does not have a closed-form likelihood.

Each dataset consists of multiple sequences of event times. The task is to predict the time τ_i until the next event given the history $\mathcal{H}(t_i)$. For each dataset, we use 60% of the sequences for training, 20% for validation and 20% for testing. We train all

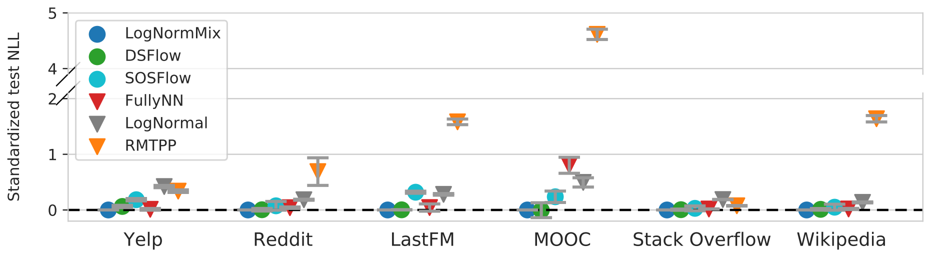


Figure 3.2: Time NLL loss when learning without marks. NLL of each model is standardized by subtracting the score of LogNormMix. **Lower score is better.** Despite its simplicity, LogNormMix consistently achieves excellent loss values.

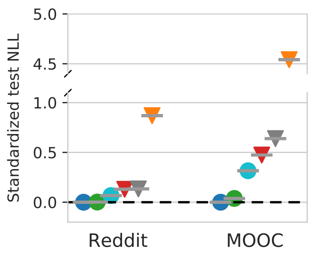


Figure 3.3: Time NLL loss when learning with marks. Rest of the setup is identical to Figure 3.2.

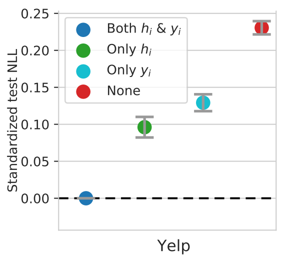
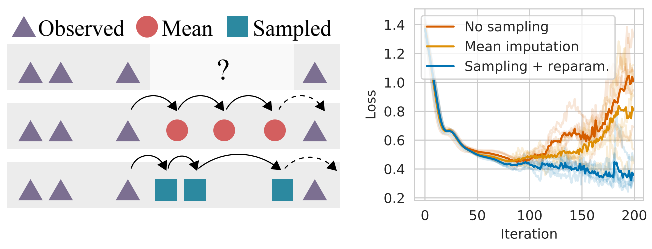


Figure 3.4: Additional conditional information improves time prediction ability of the model.

models by minimizing the negative log-likelihood (NLL) of the inter-event times in the training set. To ensure a fair comparison, we try multiple hyperparameter configurations for each model and select the best configuration using the validation set. Finally, we report the NLL loss of each model on the test set. All results are averaged over 10 train/validation/test splits. Details about the implementation, training process and hyperparameter ranges are provided in Appendix D.4. For each real-world dataset, we report the difference between the NLL loss of each method and the LogNormMix model (Figure 3.2). We report the *differences*, since scores of all models can be shifted arbitrarily by scaling the data. Absolute scores (not differences) in a tabular format, as well as results for synthetic datasets are provided in Appendix D.6.1.

Results. Simple unimodal distributions (Gompertz/RMTPP, LogNormal) are always dominated by the more flexible models with the universal approximation property (LogNormMix, DSFlow, SOSFlow, FullyNN). Among the simple models, LogNormal provides a much better fit to the data than RMTPP/Gompertz. The distribution of inter-event times in real-world data often has heavy tails, and the Gompertz distributions fails to capture this behavior. We observe that the two proposed models, LogNormMix and DSFlow consistently achieve the best loss values.



Model	NLL
No imputation	1.01 ± 0.20
Mean imputation	0.81 ± 0.27
Sampling with reparametrization	0.36 ± 0.05

Figure 3.5: By sampling the missing values from $p^*(\tau)$ during training, LogNormMix learns the true underlying data distribution. Other imputation strategies overfit the partially observed sequence.

3.4.2 Learning with marks

Setup. We apply the models for learning in *marked* temporal point processes. Marks are known to improve performance of simpler models [52], we want to establish whether our proposed models work well in this setting. We use the same setup as in the previous section, except that now each event has an associated categorical mark. In this experiment, we consider categorical marks and model them with a *conditionally independent* model (Equation 3.11). The marked model is implemented according to the description in Section 3.2.4.

Results. Figure 3.2 (right) shows the time NLL loss (i.e. $-\sum_i \log p_i^*(\tau_i)$) for Reddit and MOOC datasets. LogNormMix shows dominant performance in the marked case, just like in the previous experiment. Like before, we provide the results in tabular format, as well as report the marks NLL loss in Appendix D.6.

3.4.3 Learning with additional conditional information

Setup. We investigate whether the additional conditional information (Section 3.2.3) can improve performance of the model. In the Yelp dataset, the task is predict the time τ until the next check-in for a given restaurant. We postulate that the distribution $p_i^*(\tau)$ is different, depending on whether it is a weekday and whether it is an evening hour, and encode this information as a vector $\boldsymbol{y}_i$. We consider 4 variants of the LogNormMix model, that either use or do not use $\boldsymbol{y}_i$ and the history embedding $\boldsymbol{h}_i$.

Results. Figure 3.4 shows the test set loss for 4 variants of the model. We see that additional conditional information boosts performance of the LogNormMix model, regardless of whether the history embedding is used.

3.4.4 Missing data imputation

In practical scenarios, one often has to deal with missing data. For example, we may know that records were not kept for a period of time, or that the data is unusable for some reason. Since TPPs are a generative model, they provide a principled way to handle the missing data through imputation.

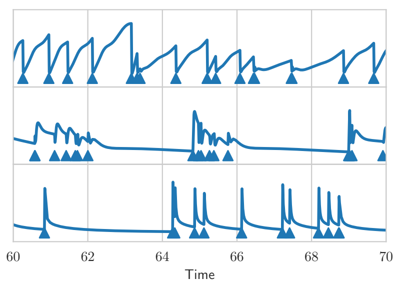


Figure 3.6: Interpolating between learned embeddings generates sequences with changing characteristics.

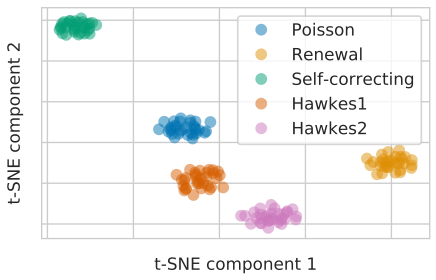


Figure 3.7: Learned embeddings let us distinguish between sequences form different distributions.

Setup. We are given several sequences generated by a Hawkes process, where some parts are known to be missing. We consider 3 strategies for learning from such a partially observed sequence: (a) ignore the gaps, maximize log-likelihood of observed inter-event times (b) fill the gaps with the average τ estimated from observed data, maximize log-likelihood of observed data, and (c) fill the gaps with samples generated by the model, maximize the *expected* log-likelihood of the observed points. The setup is demonstrated in Figure 3.5. Note that in case (c) the expected value depends on the parameters of the distribution, hence we need to perform sampling with reparametrization to optimize such loss. A more detailed description of the setup is given in Appendix D.6.4.

Results. The 3 model variants are trained on the partially-observed sequence. Figure 3.5 shows the NLL of the fully observed sequence (not seen by any model at training time) produced by each strategy. We see that strategies (a) and (b) overfit the partially observed sequence. In contrast, strategy (c) generalizes and learns the true underlying distribution. The ability of the LogNormMix model to draw samples with reparametrization was crucial to enable such training procedure.

3.4.5 Sequence embedding

Different sequences in the dataset might be generated by different processes, and exhibit different distribution of inter-event times. We can "help" the model distinguish between them by assigning a trainable embedding vector $\boldsymbol{e}_j$ to each sequence j in the dataset. It seems intuitive that embedding vectors learned this way should capture some notion of similarity between sequences.

Learned sequence embeddings. We learn a sequence embedding for each of the sequences in the synthetic datasets (along with other model parameters). We visualize the learned embeddings using t-SNE [115] in Figure 3.7 colored by the true class. As we see, the model learns to differentiate between sequences from different distributions in a completely unsupervised way.

Generation. We fit the LogNormMix model to two sequences (from self-correcting and renewal processes), and, respectively, learn two embedding vectors $\boldsymbol{e}_{SCP}$ and $\boldsymbol{e}_{RP}$. After training, we generate 3 sequences from the model, using $\boldsymbol{e}_{SCP}$, $1/2(\boldsymbol{e}_{SCP} + \boldsymbol{e}_{RP})$

and $\boldsymbol{e}_{RP}$ as sequence embeddings. Additionally, we plot the *learned* conditional intensity function of our model for each generated sequence (Figure 3.6). The model learns to map the sequence embeddings to very different distributions.

3.5 Conclusions

We use tools from neural density estimation to design new models for learning in TPPs. We show that a simple mixture model is competitive with state-of-the-art normalizing flows methods, as well as convincingly outperforms other existing approaches. By looking at learning in TPPs from a different perspective, we were able to address the shortcomings of existing intensity-based approaches, such as insufficient flexibility, lack of closed-form likelihoods and inability to generate samples analytically. We hope this alternative viewpoint will inspire new developments in the field of TPPs.

4 Fast and flexible temporal point processes with triangular maps

In Chapter 3 we presented a family of flexible neural TPPs based on autoregressive neural networks. While such models are expressive and can achieve good results in various prediction tasks, they are poorly suited for sampling — events t_i can only be generated one by one, which results in slow sequential sampling. This applies both to RNN-based models [52, 139, 167], as well as to the more recent neural TPPs based on the transformer architecture [185, 208, 217].

In this chapter we show how to overcome the above limitation and design flexible TPP models without relying on autoregressive neural networks. Our approach is based on the framework of triangular maps [91] and recent developments in the field of normalizing flows [53]. This new class of models will open new TPP applications that we will present in Chapter 5. Our main contributions are:

- We propose a new parametrization for several conventional TPPs. This enables efficient parallel likelihood computation and sampling, which was impossible with existing parametrizations.
- We propose TriTPP — a new class of TPPs that matches the flexibility of neural-network-based methods, while allowing orders of magnitude faster sampling.

4.1 Background

In this chapter we will use notation from Chapter 2 (see Appendices A and B for a recap). A TPP defines a probability distribution over variable-length sequences of strictly increasing arrival times $\mathcal{T} = (t_1, \dots, t_N)$ in a time interval $[0, T]$. One way to specify a TPP is via the conditional intensity function $\lambda^*(t) := \lambda^*(t|\mathcal{H}(t))$ that defines the rate of arrival of new events given the history $\mathcal{H}(t) = \{t_j \in \mathcal{T} : t_j < t\}$ (Equation 2.9). We will also make use of an alternative description in terms of the compensator $\Lambda^*(t) := \Lambda(t|\mathcal{H}(t))$ (Equation 2.20). We can compute the likelihood of a TPP realization $\mathcal{T}$ on $[0, T]$ as

$$\begin{aligned} p(\mathcal{T}) &= \left(\prod_{i=1}^{N} \lambda^*(t_i)\right) \exp\left(-\int_0^T \lambda^*(u) du\right) \\ &= \left(\prod_{i=1}^{N} \frac{\partial}{\partial t_i} \Lambda^*(t_i)\right) \exp\left(-\Lambda^*(T)\right). \end{aligned} \tag{4.1}$$

This quantity $p(\mathcal{T})$ is also called point process *density*.

Triangular maps [91] provide a framework that connects autoregressive models, normalizing flows and density estimation. Bogachev et al. [20] have shown that any density $p(\boldsymbol{x})$ on $\mathbb{R}^N$ can be equivalently represented by another density $\tilde{p}(\boldsymbol{z})$ on $\mathbb{R}^N$ and an increasing differentiable triangular map $\boldsymbol{F} = (f_1, \dots, f_N) : \mathbb{R}^N \to \mathbb{R}^N$ that pushes forward p into $\tilde{p}$.[1] A map $\boldsymbol{F}$ is called triangular if each component function f_i depends only on $(x_1, \dots, x_i)$ and is an increasing function of x_i. Intuitively, we can think of $\boldsymbol{F}$ as converting a random variable $\boldsymbol{x} \sim p$ into a random variable $\boldsymbol{z} := \boldsymbol{F}(\boldsymbol{x})$ with a density $\tilde{p}$. We can compute the density $p(\boldsymbol{x})$ using the change of variables formula

$$\begin{aligned} p(\boldsymbol{x}) &= |\det J_{\boldsymbol{F}}(\boldsymbol{x})| \, \tilde{p}\left(\boldsymbol{F}(\boldsymbol{x})\right) \\ &= \left(\prod_{i=1}^{N} \frac{\partial}{\partial x_i} f_i(x_1, \dots, x_i) \right) \tilde{p}\left(\boldsymbol{F}(\boldsymbol{x})\right) \end{aligned} \tag{4.2}$$

where $\det J_{\boldsymbol{F}}(\boldsymbol{x})$ is the Jacobian determinant of $\boldsymbol{F}$ at $\boldsymbol{x}$. Here, we used the fact that $J_{\boldsymbol{F}}(\boldsymbol{x})$ is a positive-definite lower-triangular matrix. To specify a complex density $p(\boldsymbol{x})$, we can pick some simple density $\tilde{p}(\boldsymbol{z})$ and learn the triangular map $\boldsymbol{F}$ that pushes p into $\tilde{p}$. It is important that $\boldsymbol{F}$ and its Jacobian determinant can be evaluated efficiently if we are learning $p(\boldsymbol{x})$ via maximum likelihood. We can sample from $p(\boldsymbol{x})$ by applying the inverse map $\boldsymbol{F}^{-1}$ to the samples drawn from $\tilde{p}(\boldsymbol{z})$. Note that $\boldsymbol{F}^{-1} : \mathbb{R}^N \to \mathbb{R}^N$ is also an increasing differentiable triangular map. Fast computation of $\boldsymbol{F}^{-1}$ is important when learning $p(\boldsymbol{x})$ via sampling-based losses (e.g., in variational inference).

4.2 Defining temporal point processes using triangular maps

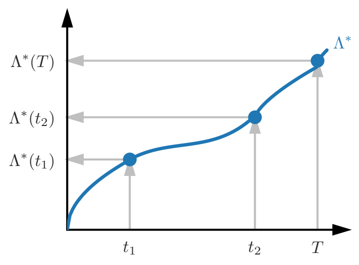
We can notice the similarity between the right-hand sides of Equations 4.1 and 4.2, which seems to suggest some connection between TPPs and triangular maps. Indeed, it turns out that triangular maps can also be used to specify densities of point processes. Let $\mathcal{T} = (t_1, \dots, t_N)$ be a realization of a TPP with compensator Λ^* on the time interval $[0, T]$ (i.e. with density $p(\mathcal{T})$). The random time change theorem (Theorem 1) states that in this case $\mathcal{Z} = (\Lambda^*(t_1), \dots, \Lambda^*(t_N))$ is a realization of the standard Poisson process (SPP) on the interval $[0, \Lambda^*(T)]$ (Figure 4.1a).

We denote the transformation that maps $\mathcal{T} = (t_1, \dots, t_N)$ to $\mathcal{Z} = (z_1, \dots, z_N)$ as $\boldsymbol{F}$

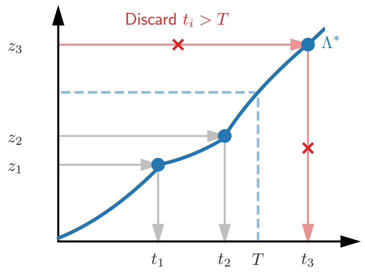
$$\boldsymbol{F}\begin{pmatrix} t_1 \\ t_2 \\ \vdots \\ t_N \end{pmatrix} = \begin{pmatrix} \Lambda(t_1) \\ \Lambda(t_2|t_1) \\ \vdots \\ \Lambda(t_N|t_1, \dots, t_{N-1}) \end{pmatrix} = \begin{pmatrix} z_1 \\ z_2 \\ \vdots \\ z_N \end{pmatrix}. \tag{4.3}$$

The transformation $\boldsymbol{F} = (f_1, \dots, f_N) \colon \mathcal{T} \mapsto \mathcal{Z}$ is indeed an increasing triangular map. Each component function $f_i(\mathcal{T}) = \Lambda(t_i|t_1, \dots, t_{i-1})$ only depends on $(t_1, \dots, t_i)$ and is increasing in t_i since $\frac{\partial}{\partial t_i}\Lambda^*(t_i) = \lambda^*(t_i) > 0$. The number N of the component functions f_i depends on the length of the specific realization $\mathcal{T}$. Notice that the term

[1]Note that some other works instead define $\boldsymbol{F}$ as the map that pushes the density $\tilde{p}(\boldsymbol{z})$ into $p(\boldsymbol{x})$.



(a) Triangular map $\boldsymbol{F}(\mathcal{T}) = (\Lambda^*(t_1), ..., \Lambda^*(t_N))$ is used for computing $p(\mathcal{T})$.



(b) Sampling is done by applying $\boldsymbol{F}^{-1}$ to a sample $\mathcal{Z}$ from the standard Poisson process.

$\prod_{i=1}^{N} \frac{\partial}{\partial t_i}\Lambda^*(t_i)$ in Equation 4.1 corresponds to the Jacobian determinant of $\boldsymbol{F}$. Similarly, the second term, $\tilde{p}(\mathcal{Z}) = \tilde{p}(\boldsymbol{F}(\mathcal{T})) = \exp(-\Lambda^*(T))$, corresponds to the density of the SPP on $[0, \Lambda^*(T)]$ for any realization $\mathcal{Z}$. This demonstrates that all TPP densities (Equation 4.1) correspond to increasing triangular maps (Equation 4.2). As for the converse of this statement, every increasing triangular map that is bijective on the space of increasing sequences defines a valid TPP (see Appendix E.1.3).

Our main idea is to define TPP densities $p(\mathcal{T})$ by directly specifying the respective maps $\boldsymbol{F}$. In Section 4.2.1, we show how maps that satisfy certain properties allow us to efficiently compute density and generate samples. We demonstrate this by designing a new parametrization for several established models in Section 4.2.2. Finally, we propose a new class of fast and flexible TPPs in Section 4.2.3.

4.2.1 Requirements for efficient TPP models

Density evaluation. The time complexity of computing the density $p(\mathcal{T})$ for various TPP models can be understood by analyzing the respective map $\boldsymbol{F}$. For a general triangular map $\boldsymbol{F} : \mathbb{R}^N \to \mathbb{R}^N$, computing $\boldsymbol{F}(\mathcal{T})$ takes $O(N^2)$ operations. For example, this holds for Hawkes processes with arbitrary kernels [79]. If the compensator Λ^* has Markov property, the complexity of evaluating $\boldsymbol{F}$ can be reduced to $O(N)$ *sequential* operations. This class of models includes Hawkes processes with exponential kernels [45, 132] and RNN-based autoregressive TPPs [52, 139, 167]. Unfortunately, such models do not benefit from the parallelism of modern hardware. Defining an efficient TPP model will require specifying a forward map $\boldsymbol{F}$ that can be computed in $O(N)$ *parallel* operations.

Sampling. As a converse of the random time change theorem, we can sample from a TPP density $p(\mathcal{T})$ by first drawing $\mathcal{Z}$ from an HPP on $[0, \Lambda^*(T)]$ and applying the inverse map, $\mathcal{T} = \boldsymbol{F}^{-1}(\mathcal{Z})$. This corresponds to the inverse transform method (Algorithm 3). There are, however, several caveats to this approach. Not all parametrizations of $\boldsymbol{F}$ allow computing $\boldsymbol{F}^{-1}(\mathcal{Z})$ in closed form. Even if $\boldsymbol{F}^{-1}$ is available, its evaluation for most models is again sequential [45, 52]. Lastly, the number of points N that will be generated (and thus $\Lambda^*(T)$ for SPP) is not known in advance. Therefore, existing methods typically resort to generating the samples one by one (Algorithm 3). We show that it is possible to do better than this. If the inverse map $\boldsymbol{F}^{-1}$ can be applied in parallel, we can produce large batches of samples t_i, and then discard the points $t_i > T$ (Figure 4.1b). Even though this method may produce samples that are later discarded, it is much more efficient than sequential generation on GPUs (Section 4.4.1).

To summarize, defining a TPP efficient for both density computation and sampling requires specifying a triangular map $\boldsymbol{F}$, such that both $\boldsymbol{F}$ and its inverse $\boldsymbol{F}^{-1}$ can be evaluated analytically in $O(N)$ *parallel* operations. We will now show that maps corresponding to several classic TPP models can be defined to satisfy these criteria.

4.2.2 Fast temporal point process models

Inhomogeneous Poisson process (IPP) (Equation 2.14) is a TPP whose conditional intensity does not depend on the history,

$$\Lambda(t|\mathcal{H}(t)) = \Lambda(t). \tag{4.4}$$

The corresponding map is $\boldsymbol{F} = \boldsymbol{\Lambda}$, where $\boldsymbol{\Lambda}$ applies the function $\Lambda : [0, T] \to \mathbb{R}_+$ elementwise to the sequence $(t_1, ..., t_N)$.

Renewal process (RP) (Equation 2.15) is a TPP where each inter-event time $t_i - t_{i-1}$ is sampled i.i.d. from the same distribution with the cumulative hazard function $\Phi\colon \mathbb{R}_+ \to \mathbb{R}_+$. The compensator of an RP is

$$\Lambda(t|\mathcal{H}(t)) = \Phi(t - t_{i-1}) + \sum_{j=1}^{i-1} \Phi(t_j - t_{j-1}), \tag{4.5}$$

where t_{i-1} is the last event before t. The triangular map of an RP can be represented as a composition $\boldsymbol{F} = \boldsymbol{C} \circ \boldsymbol{\Phi} \circ \boldsymbol{D}$, where $\boldsymbol{D} \in \mathbb{R}^{N \times N}$ is the pairwise difference matrix, $\boldsymbol{C} \equiv \boldsymbol{D}^{-1} \in \mathbb{R}^{N \times N}$ is the cumulative sum matrix, and $\boldsymbol{\Phi}$ applies Φ elementwise.

Modulated renewal process (MRP) [37] generalizes both inhomogeneous Poisson and renewal processes. The cumulative intensity is

$$\Lambda(t|\mathcal{H}(t)) = \Phi(\Lambda(t) - \Lambda(t_{i-1})) + \sum_{j=1}^{i-1} \Phi(\Lambda(t_j) - \Lambda(t_{j-1})). \tag{4.6}$$

Again, we can represent the triangular map of an MRP as a composition, $\boldsymbol{F} = \boldsymbol{C} \circ \boldsymbol{\Phi} \circ \boldsymbol{D} \circ \boldsymbol{\Lambda}$.

All three above models permit fast density evaluation and sampling. Since $\boldsymbol{\Phi}$ and $\boldsymbol{\Lambda}$ (as well as their inverses $\boldsymbol{\Phi}^{-1}$ and $\boldsymbol{\Lambda}^{-1}$) are elementwise transformations, they can obviously be applied in $O(N)$ parallel operations. Same holds for multiplication by the matrix $\boldsymbol{D}$, as it is bidiagonal. Finally, the cumulative sum defined by $\boldsymbol{C}$ can also be computed in parallel in $O(N)$ [18]. Therefore, by reformulating IPP, RP and MRP using triangular maps, we can satisfy our efficiency requirements.

Parametrization for Φ and Λ must satisfy several conditions. First, to define a valid TPP, Φ and Λ have to be positive, strictly increasing and differentiable. Next, both functions, their derivatives (for density computation) and inverses (for sampling) must be computable in closed form to meet the efficiency requirements. Lastly, we want both functions to be highly flexible. Constructing such functions is not trivial. While IPP, RP and MRP are established models, none of their existing parametrizations satisfy all the above conditions simultaneously. Luckily, the same properties are necessary when designing normalizing flows [143]. Recently, Durkan et al. [53] used rational quadratic splines (RQS) to define functions that satisfy our requirements. We propose to use RQS to define Φ and Λ for IPP, RP and MRP. This parametrization is flexible, while also allowing efficient density evaluation and sampling — something that existing approaches are unable to provide (see Section 4.3).

4.2.3 Defining more flexible triangular maps

Even though the splines can make the functions Φ and Λ arbitrarily flexible, the overall expressiveness of MRP is still limited. Its conditional intensity $\lambda^*(t)$ depends only on the global time and the time since the last event. This means, MRP cannot capture, e.g., self-exciting [79] or self-correcting [90] behavior. We will now construct a model that is more flexible without sacrificing the efficiency.

The efficiency of the MRP stems from the fact that the respective triangular map $\boldsymbol{F}$ is defined as a composition of easy-to-invert transformations. More specifically, we are combining *learnable* element-wise nonlinear transformations $\boldsymbol{\Phi}$ and $\boldsymbol{\Lambda}$ with *fixed* lower-triangular matrices $\boldsymbol{D}$ and $\boldsymbol{C}$. We can make the map $\boldsymbol{F}$ more expressive by adding *learnable* lower-triangular matrices into the composition. Using full $N \times N$ lower triangular matrices would be inefficient (multiplication and inversion are $O(N^2)$), and also would not work for variable-length sequences (i.e., arbitrary values of N). Instead, we define block-diagonal matrices $\boldsymbol{B}_l$, where each block is a repeated $H \times H$ lower-triangular matrix with strictly positive diagonal entries. Computing $\boldsymbol{B}_l^{-1}$ takes $O(H^2)$, and multiplication by $\boldsymbol{B}_l$ or $\boldsymbol{B}_l^{-1}$ can be done in $O(NH)$ in parallel. We stack L such matrices $\boldsymbol{B}_l$ and define the triangular map $\boldsymbol{F} = \boldsymbol{C} \circ \boldsymbol{\Phi}_2 \circ \boldsymbol{B}_L \circ \cdots \circ \boldsymbol{B}_1 \circ \boldsymbol{\Phi}_1 \circ \boldsymbol{D} \circ \boldsymbol{\Lambda}$. The blocks in every other layer are shifted by an offset $H/2$ to let the model capture long-range dependencies. Note that now we use two element-wise learnable splines $\boldsymbol{\Phi}_1$ and $\boldsymbol{\Phi}_2$ before and after the block-diagonal layers. Figure 4.2 visualizes the overall sequence of maps and the Jacobians of each transformation. We name the temporal point process densities defined by the triangular map $\boldsymbol{F}$ as TriTPP.

Both the forward map $\boldsymbol{F}$ and its inverse $\boldsymbol{F}^{-1}$ can be evaluated in parallel in linear time, making TriTPP efficient for density computation and sampling. Our insight that TPP densities can be represented by increasing triangular maps was crucial for arriving at this result. Alternative representations of TriTPP, e.g., in terms of the compensator Λ^* or the conditional intensity λ^*, are cumbersome and do not emphasize the parallelism of the model. TriTPP and our parametrizations of IPP, RP, MRP can be efficiently implemented on GPU to handle batches of variable-length sequences (Appendix E.1).

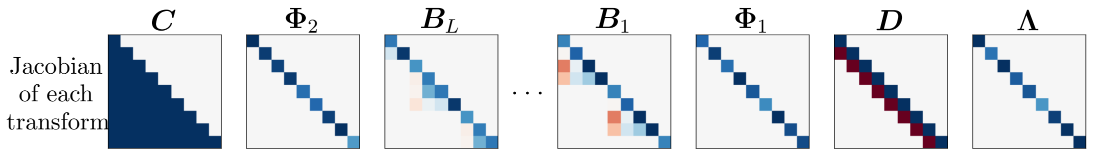


Figure 4.2: TriTPP defines an expressive map $\boldsymbol{F} = \boldsymbol{C} \circ \boldsymbol{\Phi}_2 \circ \boldsymbol{B}_L \circ \cdots \circ \boldsymbol{B}_1 \circ \boldsymbol{\Phi}_1 \circ \boldsymbol{D} \circ \boldsymbol{\Lambda}$ as a composition of easy-to-invert transformations.

4.3 Related work

Triangular maps [91] can be seen as a generalization of autoregressive normalizing flows [67, 100, 143]. Existing normalizing flow models are either limited to fixed-dimensional data [50, 142] or are inherently sequential [140, 184]. Our model proposed in Section 4.2.3 can handle variable-length inputs, and allows for both $\boldsymbol{F}$ and $\boldsymbol{F}^{-1}$ to be evaluated efficiently in parallel.

Sampling from TPPs. Inverse method for sampling from inhomogeneous Poisson processes can be dated back to Çinlar [24]. However, traditional inversion methods for IPPs are different from our approach (Section 4.2). First, they are typically sequential. Second, existing methods either use extremely basic compensators $\Lambda(t)$, such as λt or $e^{\alpha t}$, or require numerical inversion [144]. As an alternative to inversion, thinning approaches [105] became the dominant paradigm for generating IPPs, and TPPs in general. Still, sampling via thinning has a number of disadvantages. Thinning requires a piecewise-constant upper bound on $\lambda(t)$, which might not always be easy to find. If the bound is not tight, a large fraction of samples will be rejected. Moreover, thinning is not differentiable, does not permit reparametrization, and is hard to express in terms of parallel operations on tensors [181]. Our inversion-based sampling addresses all the above limitations. It is also possible to generate an IPP by first drawing $N \sim \text{Poisson}(\Lambda(T))$ and then sampling N points t_i i.i.d. from a density $p(t) = \lambda(t)/\Lambda(T)$ [36]. Unlike inversion, this method is only applicable to Poisson processes. Also, the operation of sampling N is not differentiable, which limits the utility of this approach.

Inhomogeneous Poisson processes are commonly defined by specifying the intensity function $\lambda(t)$ via a latent Gaussian process [35]. Such models are flexible, but highly intractable. It is possible to devise approximations by, e.g., bounding the intensity function [1, 51]. Our spline parametrization of IPP compares favorably to the above models: it is also highly flexible, has a tractable likelihood and places no restrictions on the intensity. Importantly, it is much easier to implement and train. If uncertainty is of interest, we can perform approximate Bayesian inference on the spline coefficients [207]. Recently, Morgan et al. [124] used splines to model the intensity function of IPPs. Since Λ^{-1} cannot be computed analytically for their model, sampling via thinning is the only available option.

Modulated renewal processes have been known for a long time [12, 37], but have not become as popular as IPPs among practitioners. This is not surprising, since inference and sampling in MRPs are even more challenging than in Cox processes [103, 150].

Our proposed parametrization addresses the shortcomings of existing approaches and makes MRPs straightforward to apply in practice.

Neural TPPs. Du et al. [52] proposed a TPP model based on a recurrent neural network. Follow-up works improved the flexibility of RNN-based TPPs by e.g. changing the RNN architecture [119], using more expressive conditional hazard functions [26, 139] or modeling the inter-event time distribution with normalizing flows [167]. All the above models are inherently sequential and therefore inefficient for sampling (Section 4.4.1). Turkmen et al. [181] proposed to speed up RNN-based *marked* TPPs by discretizing the interval $[0, T]$ into a regular grid. Samples within each grid cell can be produced in parallel for each mark, but the cells themselves still must be processed sequentially.

Several recent neural TPP models replaced the RNN with a transformer neural network [208, 215, 217]. Unlike RNN-based TPPs, such models can compute the likelihood in parallel. However, the time and memory complexity of transformer models scales as $O(N^2)$, which limits their applicability to sequences with more than 10^3 events [185]. More importantly, these models are only able to generate events one by one because of their autoregressive structure. Quadratic scaling and sequential dependencies mean that, in practice, transformer-based TPPs are even less scalable than their RNN-based counterparts.

4.4 Experiments

4.4.1 Scalability

Setup. The key feature of TriTPP is its ability to compute likelihood and generate samples in parallel, which is impossible for RNN-based models. We quantify this difference by measuring the runtime of the two models. We implemented TriTPP and RNN models in PyTorch [145]. The architecture of the RNN model is nearly identical to the ones used in [52, 139, 167], except that the cumulative conditional hazard function is parametrized with a spline [53] to enable closed-form sampling. Appendix E.3 contains the details for this and other experiments. We measure the runtime of (a) computing the log-likelihood (and backpropagate the gradients) for a batch of 100 sequences of varying lengths and (b) sample sequences of the same sizes. We used a machine with an Intel Xeon E5-2630 v4 @ 2.20 GHz CPU, 256GB RAM and an Nvidia GTX1080Ti GPU. The results are averaged over 100 runs.

Results. Figure 4.3 shows the runtimes for varying sequence lengths. Training is rather fast for both models, on average taking 1-10ms per iteration. RNN is slightly faster for short sequences, but is outperformed by TriTPP on sequences with more than 400 events. Note that during training we used a highly optimized RNN implementation based on custom CUDA kernels (since all the event times t_i are already known). In contrast, TriTPP is implemented using generic PyTorch operations. When it comes to sampling, we notice a massive gap in performance between TriTPP and the RNN model. This happens because RNN-based TPPs are defined autoregressively and can only produce samples t_i one by one: to obtain $p(t_i|t_1, ..., t_{i-1})$ we must know all the past events. Recently proposed transformer TPPs [208, 217] are defined in a similar

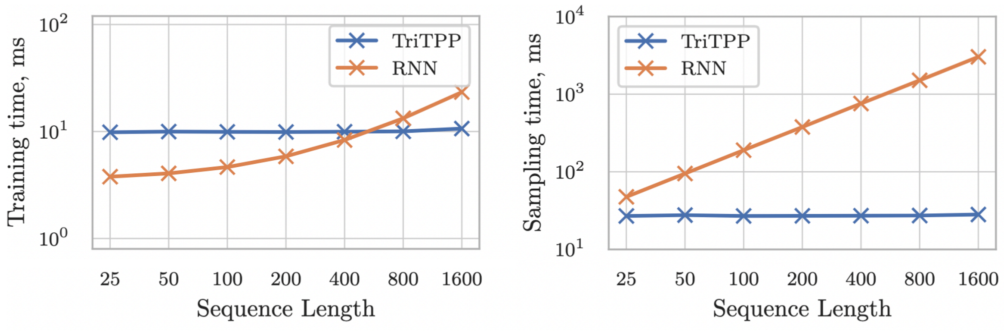


Figure 4.3: Scalability analysis. Standard deviations are below 1ms.

autoregressive way, so they are likely to be as slow for sampling as RNNs. TriTPP generates all the events in a sequence in parallel, which makes it more than 100 times faster than the recurrent model for longer sequences.

4.4.2 Density estimation

Setup. A fast TPP model is of little use if it cannot accurately learn the data distribution. The main goal of this experiment is to establish whether TriTPP can match the flexibility of RNN-based TPPs. As baselines, we use the IPP, RP and MRP models from Section 4.2.2 and Hawkes process [8].

Datasets. We use 6 synthetic datasets from Omi et al. [139]: Hawkes1&2 [79], self-correcting (SCP) [90], inhomogeneous Poisson (IPP), renewal (RP) and modulated renewal (MRP) processes. Note that the data generators for IPP, RP and MRP by Omi et al. are *not* parametrized using splines, so these datasets are not guaranteed to be fitted perfectly by our models. We also consider 7 real-world datasets: PUBG (online gaming), Reddit-Comments, Reddit-Submissions (online discussions), Taxi (customer pickups), Twitter (tweets) and Yelp1&2 (check-in times). See Appendix E.2 for more details.

Metrics. The standard metric for comparing generative models, including TPPs, is negative log-likelihood (NLL) on a hold-out set [139, 167, 181]. We partitioned the sequences in each dataset into train/validation/test sequences (60%/20%/20%). We trained the models by minimizing the NLL of the train set using Adam [98]. We tuned the following hyperparameters: L_2 regularization $\{0, 10^{-5}, 10^{-4}, 10^{-3}\}$, number of spline knots $\{10, 20, 50\}$, learning rate $\{10^{-3}, 10^{-2}\}$, hidden size $\{32, 64\}$ for RNN, number of blocks $\{2, 4\}$ and block size $\{8, 16\}$ for TriTPP. We used the validaiton set for hyperparameter tuning, early stopping and model development. We computed the results for the test set only once before including them in the paper. All results are averaged over 5 runs.

While NLL is a popular metric, it has known failure modes [180]. For this reason, we additionally computed maximum mean discrepancy (MMD) [76] between the test sets and the samples drawn from each model after training. To measure similarity between two realizations $\mathcal{T}$ and $\mathcal{U}$, we use a Gaussian kernel $k(\mathcal{T}, \mathcal{U}) = \exp(-d(\mathcal{T}, \mathcal{U})/2\sigma^2)$, where

Table 4.1: Average test set NLL on synthetic and real-world datasets (lower is better). Best NLL in **bold**, second best underlined.

	Hawkes1	Hawkes2	SC	IPP	MRP	RP	PUBG	Reddit-C	Reddit-S	Taxi	Twitter	Yelp1	Yelp2
IPP	1.06	1.03	1.00	**0.71**	0.70	0.89	-0.06	-1.59	-4.08	**-0.68**	1.60	0.62	-0.05
RP	0.65	0.08	0.94	0.85	0.68	**0.24**	0.12	-2.08	-4.00	-0.58	1.20	0.67	-0.02
MRP	0.65	0.07	0.93	**0.71**	0.36	0.25	-0.83	-2.13	-4.38	**-0.68**	1.23	**0.61**	**-0.10**
Hawkes	**0.51**	0.06	1.00	0.86	0.98	0.39	0.11	**-2.40**	-4.19	-0.64	**1.04**	0.69	0.01
RNN	0.52	**-0.03**	**0.79**	0.73	0.37	**0.24**	-1.96	**-2.40**	**-4.89**	-0.66	1.08	0.67	-0.08
TriTPP	0.56	0.00	0.83	**0.71**	**0.35**	**0.24**	**-2.41**	-2.36	-4.49	-0.67	1.06	0.64	-0.09

Table 4.2: MMD between the hold-out test set and the generated samples (lower is better).

	Hawkes1	Hawkes2	SC	IPP	MRP	RP	PUBG	Reddit-C	Reddit-S	Taxi	Twitter	Yelp1	Yelp2
IPP	0.08	0.09	0.58	**0.02**	0.15	0.07	**0.01**	0.10	0.21	0.10	0.16	0.15	0.16
RP	0.06	0.06	1.13	0.34	1.24	**0.01**	0.46	0.07	0.18	0.57	0.14	0.16	0.23
MRP	0.05	0.06	0.50	**0.02**	0.11	0.02	0.12	0.09	0.20	0.09	0.13	0.13	0.16
Hawkes	0.02	0.04	0.58	0.36	0.65	0.05	0.16	**0.04**	0.35	0.20	0.20	0.20	0.32
RNN	**0.01**	**0.02**	**0.19**	0.09	0.17	**0.01**	0.23	**0.04**	**0.09**	0.13	**0.08**	0.19	0.18
TriTPP	0.03	0.03	0.23	**0.02**	**0.08**	**0.01**	0.16	0.07	0.16	**0.08**	**0.08**	**0.12**	**0.14**

$d(\mathcal{T},\mathcal{U})$ is the "counting measure" distance from [199, Equation 3]. For completeness, we provide the definitions in Appendix E.3.2. MMD quantifies the dissimilarity between the true data distribution $p^\star(\mathcal{T})$ and the learned density $p(\mathcal{T})$ — lower is better.

Results. Table 4.1 shows the test set NLLs for all models and datasets. We can see that the RNN model achieves excellent scores and outperforms the simpler baselines, which is consistent with earlier findings [52]. TriTPP is the only method that is competitive with the RNN — our method is within 0.05 nats of the best score on 11 out of 13 datasets. TriTPP consistently beats MRP, RP and IPP, which confirms that learnable block-diagonal transformations improve the flexibility of the model. The gap get larger on the datasets such as Hawkes, SCP, PUBG and Twitter, where the inability of MRP to learn self-exciting and self-correcting behavior is especially detrimental. While Hawkes process is able to achieve good scores on datasets with "bursty" event occurrences (Reddit, Twitter), it is unable to adequately model other types of behavior (SCP, MRP, PUBG).

Table 4.2 reports the MMD scores. The results are consistent with the previous experiment: models with lower NLL typically obtain lower MMD. One exception is the Hawkes process that achieves low NLL but high MMD on Taxi and Twitter. TriTPP again consistently demonstrates excellent performance. Note that MMD was computed using the test sequences that were unseen during training. This means that TriTPP models the data distribution better than other methods, and does not just simply overfit the training set. In Appendix E.4.1, we provide additional experiments for quantifying the quality of the distributions learned by different models. Overall, we conclude that TriTPP is flexible and able to model complex densities, in addition to being significantly more efficient than RNN-based TPPs.

4.5 Conclusions

Future work & limitations. We parametrized the nonlinear transformations of our TPP models with splines. Making a spline more flexible requires increasing the number of knots, which increases the number of parameters and might lead to overfitting. New deep *analytically invertible* functions will improve both our models, as well as normalizing flows in general. Currently, TriTPP is not applicable to marked TPPs [152]. Extending our model to this setting is an important task for future work.

Conclusions. We have shown that TPP densities can be represented with increasing triangular maps. By directly parametrizing the respective transformations, we are able to construct TPP models, for which both density evaluation and sampling can be done efficiently in parallel. Using the above framework, we defined TriTPP— a new class of flexible probability distributions over variable-length sequences. In addition to being highly efficient thanks to its parallelism, TriTPP shows excellent performance on density estimation, as shown by our experiments. High flexibility and efficiency of TriTPP make it perfectly suited for tasks beyond density estimation, as we will demonstrate in the next chapter.

Part III

Applications

5 Learning with sampling-based losses

In Chapters 3 and 4 we introduced flexible neural TPP models, where new event sequences can be sampled exactly using the inversion method (Algorithm 3). The ability to sample from a TPP allows us to answer prediction queries such as "How many events are expected to happen in the next hour given the history?". Even more importantly, sampling *with reparametrization* enables us to train TPPs with sampling-based losses that arise in application areas such as reinforcement learning [108, 183], variational inference [28, 168], and adversarial training [198, 199, 202].

Unfortunately, as we demonstrate in this chapter, such sampling-based losses for TPPs are plagued by discontinuities, which prevents us from effectively optimizing them with gradient descent. We propose a solution to this problem by introducing a differentiable relaxation for TPP losses. To demonstrate the utility of this approach, we derive a variational inference scheme for continuous-time discrete-state probabilistic models known as Markov jump processes [141]. To summarize, our contributions are the following:

- We introduce the reparametrization trick for TPPs. Combined with a new differentiable relaxation for TPP losses, it allows us to efficiently train TPPs using sampling-based objective functions.
- Based on the previous insight, we develop a variational inference scheme for Markov jump processes.

5.1 Background

5.1.1 Sampling-based losses for TPPs

We can think of a TPP as a probability distribution $p_{\boldsymbol{\theta}}(\mathcal{T})$ over variable-length event sequences $\mathcal{T} = (t_1, \dots, t_N)$. In Chapters 2 to 4 we focused on learning TPP parameters $\boldsymbol{\theta}$ using maximum likelihood estimation (Equation 2.25).

$$\max_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\mathcal{T}). \tag{5.1}$$

This, however, is not the only possible option. In a number applications we are interested in learning TPP models using objective functions involving expectations

$$\max_{\boldsymbol{\theta}} \mathbb{E}_{\mathcal{T} \sim p_{\boldsymbol{\theta}}(\mathcal{T})}[g(\mathcal{T})]. \tag{5.2}$$

We refer to such objectives as *sampling-based losses*, since they involve averaging over event sequences $\mathcal{T}$ that are sampled from the TPP $p_{\boldsymbol{\theta}}(\mathcal{T})$. Let us consider several examples to make this discussion more concrete.

Reinforcement learning. Upadhyay et al. [183] apply TPPs to find optimal times for posting content on social media to get the highest possible ranking in the feed. We can think of the times $\mathcal{T} = (t_1, \dots, t_N)$ when content is posted as a TPP realization, and therefore model our stochastic policy that chooses when to make posts with a TPP $p(\mathcal{T})$. The goal is to maximize the reward $r(\mathcal{T})$ that is defined as the fraction of the time spent on top of the followers' news feeds. This is equivalent to finding a policy $p(\mathcal{T})$, defined by a TPP, that that maximizes the following objective function

$$\max_p \; \mathbb{E}_{\mathcal{T} \sim p(\mathcal{T})}[r(\mathcal{T})]. \tag{5.3}$$

Variational inference. Consider a system that is governed by a binary latent state that switches at random times [61]. For example, a computer in a network might behave differently depending on whether it is in "on" or "off" state. We can model the switching times $\mathcal{T} = (t_1, \dots, t_N)$ as a latent variable with prior distribution $p(\mathcal{T})$. The switching times $\mathcal{T}$ determine the state of the system, which in turn affects the observed behavior denoted as $\mathcal{X}$. We represent this mechanism through a conditional distribution $p(\mathcal{X}|\mathcal{T})$.

A common task is to infer the unobserved switching times $\mathcal{T}$ given the observations $\mathcal{X}$. Unfortunately, the posterior distribution $p(\mathcal{T}|\mathcal{X})$ is often intractable. The main idea of variation inference is to approximate the intractable posterior $p(\mathcal{T}|\mathcal{X})$ with a tractable distribution $q(\mathcal{T})$ [17, 187]. We can find the best possible approximation by minimizing the Kullback–Leibler divergence between the approximate posterior $q(\mathcal{T})$ and the true posterior $p(\mathcal{T}|\mathcal{X})$, which corresponds to maximizing the Evidence Lower BOund (ELBO)

$$\max_q \; \mathbb{E}_{\mathcal{T} \sim q(\mathcal{T})}[\log p(\mathcal{X}|\mathcal{T}) + \log p(\mathcal{T}) - \log q(\mathcal{T})]. \tag{5.4}$$

In the case that we consider, $\mathcal{T}$ is a variable-length sequence of events in continuous time, so $p(\mathcal{T})$, $p(\mathcal{T}|\mathcal{X})$ and $q(\mathcal{T})$ are all represented by TPPs.

Adversarial learning provides an alternative to the maximum likelihood parameter estimation procedure for generative models. Suppose we would like to fit a TPP $p(\mathcal{T})$ using a training set $\mathcal{D}_{\text{train}} = \{\mathcal{T}^{(1)}, \dots, \mathcal{T}^{(M)}\}$. The main idea of adversarial training is to additionally train a discriminator function $d(\cdot)$ that should discriminate between true instances from $\mathcal{D}_{\text{train}}$ and sequences sampled from $p(\mathcal{T})$ [198, 199, 202]. This corresponds to solving the following optimization problem

$$\max_p \; \min_d \; \mathbb{E}_{\mathcal{T} \sim p(\mathcal{T})}\left[d(\mathcal{T})\right] - \frac{1}{M} \sum_{m=1}^{M} d\left(\mathcal{T}^{(m)}\right). \tag{5.5}$$

Optimizing with respect to both p and d (subject to certain constraints [6]) forces the generative model $p(\mathcal{T})$ closer to the distribution that produced $\mathcal{D}_{\text{train}}$.

5.1.2 Monte Carlo gradient estimators

The standard way to solve optimization problems such as in Equation 5.3–5.5 is to pick a parametric TPP model $p_{\boldsymbol{\theta}}(\mathcal{T})$ and to learn its parameters $\boldsymbol{\theta}$ with a gradient-based method. For this we need to be able to evaluate the gradient of the objective function

$$\nabla_{\boldsymbol{\theta}} \, \mathbb{E}_{\mathcal{T} \sim p_{\boldsymbol{\theta}}(\mathcal{T})}[g(\mathcal{T})]. \tag{5.6}$$

In all but simplest cases, neither the expectation $\mathbb{E}_{\mathcal{T} \sim p_{\boldsymbol{\theta}}(\mathcal{T})}[g(\mathcal{T})]$, nor its gradient w.r.t. $\boldsymbol{\theta}$ can be computed analytically. We can, however, approximate Equation 5.6 using approaches known as Monte Carlo (MC) gradient estimators [121].

Score function estimator [196] (also known as REINFORCE) is one example of MC gradient estimators. This approach is based on the following identity

$$\nabla_{\boldsymbol{\theta}} \mathbb{E}_{\mathcal{T} \sim p_{\boldsymbol{\theta}}(\mathcal{T})}[g(\mathcal{T})] = \mathbb{E}_{\mathcal{T} \sim p_{\boldsymbol{\theta}}(\mathcal{T})}[\nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\mathcal{T}) g(\mathcal{T})]. \tag{5.7}$$

We can approximate the expectation using samples $\mathcal{T}^{(1)}, \dots, \mathcal{T}^{(S)}$ drawn from $p_{\boldsymbol{\theta}}(\mathcal{T})$ as

$$\mathbb{E}_{\mathcal{T} \sim p_{\boldsymbol{\theta}}(\mathcal{T})}[\nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\mathcal{T}) g(\mathcal{T})] \approx \frac{1}{S} \sum_{s=1}^{S} \nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}} \left(\mathcal{T}^{(s)}\right) g\left(\mathcal{T}^{(s)}\right). \tag{5.8}$$

This is a generic approach that can be combined with any generative model, where sampling and density computation are tractable. This algorithm has also been used in the context of TPPs [108, 183]. The main disadvantage of the score function estimator is its high variance [121], so applying the estimator in practice often requires employing additional variance reduction techniques [147].

Reparametrization trick (also known as the pathwise gradient estimator) [99, 157] provides a lower-variance alternative to the score function estimator [121]. The main idea of this approach is to replace sampling from $p_{\boldsymbol{\theta}}(\mathcal{T})$ by the following two-step procedure. First, we sample $\mathcal{Z}$ from a distribution $\tilde{p}(\mathcal{Z})$ that does not depend on $\boldsymbol{\theta}$. Then, we obtain $\mathcal{T}$ by passing $\mathcal{Z}$ through a *deterministic* transformation parametrized by $\boldsymbol{\theta}$. We will now show how this approach can be generalized to TPPs.

5.2 Reparametrization trick for TPPs

5.2.1 Inversion method as reparametrization sampling

In Chapter 4, we discussed how sampling from any TPP $p_{\boldsymbol{\theta}}(\mathcal{T})$ can be represented using the same two-step procedure: We generate $\mathcal{Z}$ from the standard Poisson process $\tilde{p}(\mathcal{Z})$ (Equation 2.13) and then apply the triangular map $\boldsymbol{F}_{\boldsymbol{\theta}}^{-1}$ to obtain a sample from $p_{\boldsymbol{\theta}}(\mathcal{T})$ as $\mathcal{T} = \boldsymbol{F}_{\boldsymbol{\theta}}^{-1}(\mathcal{Z})$. This corresponds to the reparametrization trick and, therefore, allows us to compute gradients of sampling-based losses

$$\nabla_{\boldsymbol{\theta}} \mathbb{E}_{\mathcal{T} \sim p_{\boldsymbol{\theta}}(\mathcal{T})}[g(\mathcal{T})] = \mathbb{E}_{\mathcal{Z} \sim \tilde{p}(\mathcal{Z})} \left[\nabla_{\boldsymbol{\theta}} g\left(\boldsymbol{F}_{\boldsymbol{\theta}}^{-1}(\mathcal{Z})\right)\right]. \tag{5.9}$$

We approximate this expectation using samples $\mathcal{Z}^{(1)}, \dots, \mathcal{Z}^{(S)}$ drawn from the SPP $\tilde{p}(\mathcal{Z})$

$$\mathbb{E}_{\mathcal{Z} \sim \tilde{p}(\mathcal{Z})} \left[\nabla_{\boldsymbol{\theta}} g(\boldsymbol{F}_{\boldsymbol{\theta}}^{-1}(\mathcal{Z}))\right] \approx \frac{1}{S} \sum_{s=1}^{S} g\left(\boldsymbol{F}_{\boldsymbol{\theta}}^{-1}\left(\mathcal{Z}^{(s)}\right)\right). \tag{5.10}$$

The main prerequisite for applying the above technique is the ability to compute the inverse map $\boldsymbol{F}_{\boldsymbol{\theta}}^{-1}$ analytically. This is true for the models that we developed in Chapter 4. TriTPP (Section 4.2.3) defines a flexible triangular map $\boldsymbol{F}_{\boldsymbol{\theta}}^{-1}$ that can be applied in $O(N)$ parallel operations. This means, TriTPP is perfectly suited for learning with sampling-based objectives — sampling is efficient thanks to the parallelism of the map $\boldsymbol{F}_{\boldsymbol{\theta}}^{-1}$, and the reparametrization trick allows us to compute gradients with respect to the TPP parameters $\boldsymbol{\theta}$. The same applies to parametrizations of inhomogeneous Poisson and (modulated) renewal processes that we introduced in Section 4.2.2. Finally, the LogNormMix model from Chapter 3 also permits approximate reparametrization sampling, however, not as efficiently due to sequential dependencies in the RNN (see Section 3.2.2 for details).

Non-differentiability of TPP losses. The reparametrization trick for TPPs (Equations 5.9 and 5.10) allows us to compute gradients of sampling-based losses w.r.t. the TPP parameters $\boldsymbol{\theta}$. Unfortunately, this is not sufficient to optimize such losses with gradient-based methods — as we will see shortly, such losses for TPPs are in general discontinuous and, therefore, not differentiable. This is a property of the loss functions that is independent of the parametrization of $p_{\boldsymbol{\theta}}(\mathcal{T})$. In the following, we provide a simple example and a solution to this problem.

5.2.2 Differentiable relaxation for TPP losses

Entropy maximization. To demonstrate the non-differentiability problem, we consider a toy task of maximizing the entropy of a homogeneous Poisson process. An entropy penalty can be used as a regularizer during density estimation [73] or as a part of the ELBO in variational inference [187].

Let $p_\mu(\mathcal{T})$ be a homogeneous Poisson process (HPP) on $[0, T]$ with rate $\mu > 0$. It is known that the entropy is maximized when $\mu = 1$, but for sake of example assume that we want to learn μ that maximizes the entropy with gradient ascent [7]. This corresponds to the following optimization problem

$$\max_{\mu \in \mathbb{R}_+} \mathbb{E}_{\mathcal{T} \sim p_\mu(\mathcal{T})}[-\log p_\mu(\mathcal{T})]. \tag{5.11}$$

We use the reparametrization trick for TPPs to estimate the objective function. We generate a sequence $\mathcal{Z} = (z_1, z_2, ...)$ from the standard Poisson process (SPP) and apply the inverse map $\mathcal{T} = \boldsymbol{F}_\mu^{-1}(\mathcal{Z}) = \frac{1}{\mu}\mathcal{Z}$. We obtain an Monte Carlo estimate of the entropy using a single such sample $\mathcal{T} = (t_1, t_2, ...)$ as

$$\begin{aligned} \mathbb{E}_{\mathcal{T} \sim p_\mu(\mathcal{T})}[-\log p_\mu(\mathcal{T})] &\approx \mu T - \sum_{i=1}^{\infty} \mathbb{1}(t_i \le T) \log \mu \\ &= \mu T - \sum_{i=1}^{\infty} \mathbb{1}\left(\frac{1}{\mu} z_i \le T\right) \log \mu \end{aligned} \tag{5.12}$$

Here, the indicator function $\mathbb{1}(\cdot)$ discards all the events $t_i > T$.

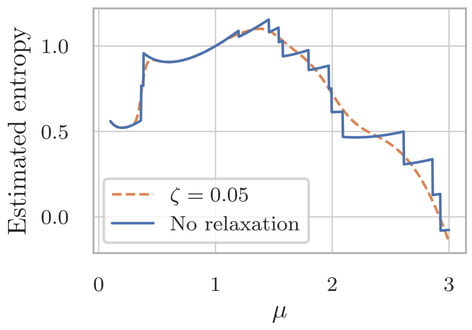


Figure 5.1: Monte Carlo estimate of the entropy.

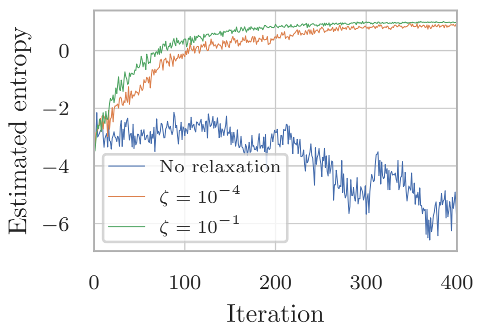


Figure 5.2: Maximizing the entropy with different values of ζ.

We can see that for any sample $\mathcal{Z} = (z_1, z_2, \dots)$ the right-hand side of Equation 5.12 is not continuous with respect to μ at points $\mu = \frac{1}{T} z_i$. At such points, decreasing μ by an infinitesimal amount will "push" the event $t_i = \frac{1}{\mu} z_i$ outside the $[0, T]$ interval, thus increasing $\log p_\mu(\mathcal{T})$ by a constant $\log \mu$. We plot the right-hand side of Equation 5.12 as a function of μ in Figure 5.1, estimated with 5 MC samples. Clearly, such a function cannot be optimized with gradient ascent. Increasing the number of MC samples almost surely adds more points of discontinuity and does not fix the problem. In general, non-differentiability arises when estimating expectations of a function $g(\mathcal{T})$ that depends on the events t_i inside $[0, T]$. For any TPP density $p(\mathcal{T})$, the discontinuities occur at the parameter values that map the SPP events z_i to the interval boundary T.

Relaxation. We obtain a differentiable approximation to Equation 5.12 by replacing the indicator functions $\mathbb{1}(t_i \le T)$ with sigmoid $\sigma_\zeta(T - t_i)$ as

$$\mu T - \sum_{i=1}^{\infty} \mathbb{1}\left(\frac{1}{\mu} z_i \le T\right) \log \mu \approx \mu T - \sum_{i=1}^{\infty} \sigma_\zeta \left(T - \frac{1}{\mu} z_i\right) \log \mu, \tag{5.13}$$

where $\sigma_\zeta(x) = 1/(1 + \exp(-x/\zeta))$ is the sigmoid function with a temperature parameter $\zeta > 0$. This relaxation is similar in spirit to the Gumbel-softmax trick for the categorical distribution [92, 116] — we enable reparametrization gradients and decrease the variance at the cost of introducing bias to the gradient estimates. Decreasing the temperature ζ makes the approximation more accurate but complicates optimization, and $\zeta = 0$ recovers the original non-differentiable objective.

Figure 5.2 shows the convergence plots of the entropy maximization task for different temperature values ζ. As expected, gradient ascent fails on the original non-differentiable objective function but works well on the relaxed objective. This example demonstrates the feasibility of optimizing sampling-based TPP losses using reparametrization sampling and our differentiable relaxation technique.

Next, we compare the variance of the relaxed reparametrization gradient estimator (Equation 5.10) to the variance of the score function estimator (Equation 5.8). Figure 5.3 shows the variance of the different estimators as a function of rate parameter μ. We

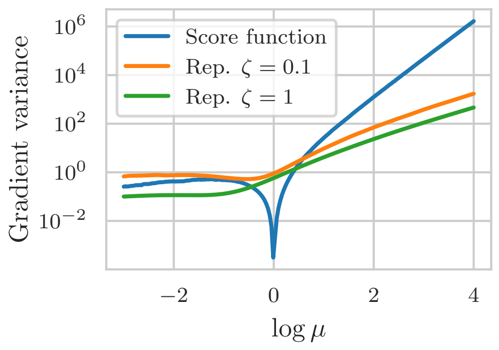


Figure 5.3: Gradient variance for score function and reparametrization gradient estimators in the HPP entropy maximization task.

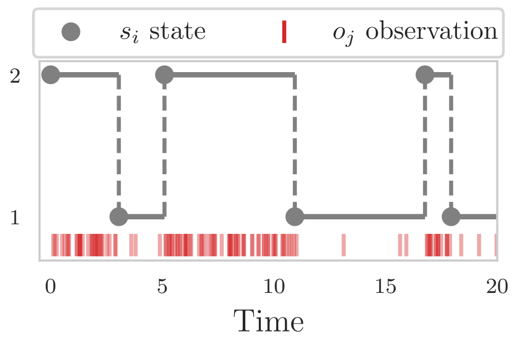


Figure 5.4: Markov modulated Poisson process. Latent state follows a 2-state MJP that controls the arrival rate of observed events.

observe that the reparametrization estimator achieves lower variance than the score function estimator, especially at larger values of μ (note the log scale). This follows the general trend observed in other application areas [121] and highlights the advantages of the reparametrization trick over the score function estimator.

Summary. We showed how sampling-based TPP losses can be optimized using the reparametrization trick. Our approach consists of two components: inverse transform sampling allows us to compute the gradient with respect to model parameters, and the differentiable relaxation makes gradient-based optimization possible. Our relaxation scheme is applicable to any TPP loss $g(\mathcal{T})$ that can be expressed in terms of the indicator functions. In the next section, we will combine this framework with our flexible and efficient models from Chapter 4 to develop a variational inference scheme for Markov jump processes.

5.3 Variational inference for Markov jump processes

Background. A Markov jump process (MJP) $\{s(t)\}_{t \geq 0}$ is a piecewise-constant stochastic process on $[0, \infty)$. At any time t, the process occupies a discrete state $s(t) \in \{1, ..., K\}$. The times when the state changes are called jumps. A trajectory of an MJP on an interval $[0, T]$ with N jumps can be represented by a tuple $(\mathcal{T}, \mathcal{S})$ of jump times $\mathcal{T} = (t_1, ..., t_N)$ and the visited states $\mathcal{S} = (s_1, ..., s_{N+1})$. Note that N may vary for different trajectories. The prior over the trajectories $p(\mathcal{T}, \mathcal{S} | \boldsymbol{\pi}, \boldsymbol{A})$ of an MJP is governed by an initial state distribution $\boldsymbol{\pi}$ and a $K \times K$ generator matrix $\boldsymbol{A}$ (see Appendix F.1.1).

MJPs are commonly used to model the unobserved (latent) state of a system. In a latent MJP, the state $s(t)$ influences the behavior of the system and indirectly manifests itself via some observations $\mathcal{X}$. For concreteness, we consider the Markov-modulated Poisson process (MMPP) [61]. In an MMPP, each of the K states of the MJP has an associated observation intensity λ_k. An MMPP is an inhomogeneous Poisson process where the intensity depends on the current MJP state as $\lambda^*(t) = \lambda_{s(t)}$. For instance, a 2-state MMPP can model the behavior of a social network user, who switches between an "active" (posting a lot) and "inactive" (working or sleeping) states (Figure 5.4). Given the observations $\mathcal{X}$, we might be interested in inferring the trajectory $(\mathcal{T}, \mathcal{S})$, the model parameters $\boldsymbol{\psi} = \{\boldsymbol{\pi}, \boldsymbol{A}, \boldsymbol{\lambda}\}$, or both.

Variational inference (VI). The posterior distribution $p(\mathcal{T}, \mathcal{S}|\mathcal{X}, \boldsymbol{\psi})$ of MMPP is intractable, so we approximate it with a variational distribution $q(\mathcal{T}, \mathcal{S}) = q(\mathcal{T})q(\mathcal{S}|\mathcal{T})$. Note that this is *not* a mean-field approximation used in other works [206]. We model the distribution over the jump times $q(\mathcal{T})$ with TriTPP (Section 4.2.3). We find the best approximate posterior by maximizing the ELBO [207]

$$\max_{q(\mathcal{T})} \max_{q(\mathcal{S}|\mathcal{T})} \mathbb{E}_{q(\mathcal{T})} \left[\mathbb{E}_{q(\mathcal{S}|\mathcal{T})} \left[\log p(\mathcal{X}|\mathcal{T}, \mathcal{S}, \boldsymbol{\psi}) + \log p(\mathcal{T}, \mathcal{S}|\boldsymbol{\psi}) - \log q(\mathcal{T}, \mathcal{S})\right]\right] \tag{5.14}$$

Given jump times $\mathcal{T}$, the true posterior over the states $p(\mathcal{S}|\mathcal{T}, \mathcal{X}, \boldsymbol{\psi})$ is just the posterior of a discrete hidden Markov model (HMM). This means we only need to model $q(\mathcal{T})$; the optimal posterior over the states $q^\star(\mathcal{S}|\mathcal{T})$

$$\begin{aligned} q^\star(\mathcal{S}|\mathcal{T}) &= \operatorname*{arg\,max}_{q(\mathcal{S}|\mathcal{T})} \mathbb{E}_{q(\mathcal{S}|\mathcal{T})} \left[\log p(\mathcal{X}|\mathcal{T}, \mathcal{S}, \boldsymbol{\psi}) + \log p(\mathcal{T}, \mathcal{S}|\boldsymbol{\psi}) - \log q(\mathcal{S}|\mathcal{T})\right] && (5.15)\\ &= p(\mathcal{S}|\mathcal{T}, \mathcal{X}, \boldsymbol{\psi}) && (5.16) \end{aligned}$$

can be found efficiently via the forward-backward algorithm [188]. The inner expectation w.r.t. $q(\mathcal{S}|\mathcal{T})$ in Equation 5.14 can be computed analytically. We approximate the expectation w.r.t. $q(\mathcal{T})$ with Monte Carlo. Since all terms of Equation 5.14 are not differentiable, we apply our relaxation from Section 5.2.2. We provide a full derivation of the ELBO and the implementation details in Appendix F.1.3.

The proposed framework is not limited to approximating the posterior over the trajectories. With small modifications (Appendix F.1.4), we can simultaneously learn the parameters $\boldsymbol{\psi}$, either obtaining a point estimate $\boldsymbol{\psi}^\star$ or a full approximate posterior $q(\boldsymbol{\psi})$. Our variational inference scheme can also be extended to other continuous-time discrete-state models, such as semi-Markov processes [59].

5.4 Related work

Monte Carlo gradient estimators play an important role both in machine learning [121] as well as other research fields [33, 160]. Score function gradient estimator [196] and the reparametrization trick [99, 157] are the two families of approaches most commonly used in practice. Reparametrization trick typically provides lower variance and therefore faster convergence, but is not as general as the score function estimator [121]. For example, exact reparametrization sampling from discrete distributions is not possible in general. However, there exist relaxation methods that allow us to approximate reparametrization sampling, e.g., for the categorical distribution [92, 116]. Our relaxation technique complements these approaches and can be seen as their extension to event sequences with a random number of events.

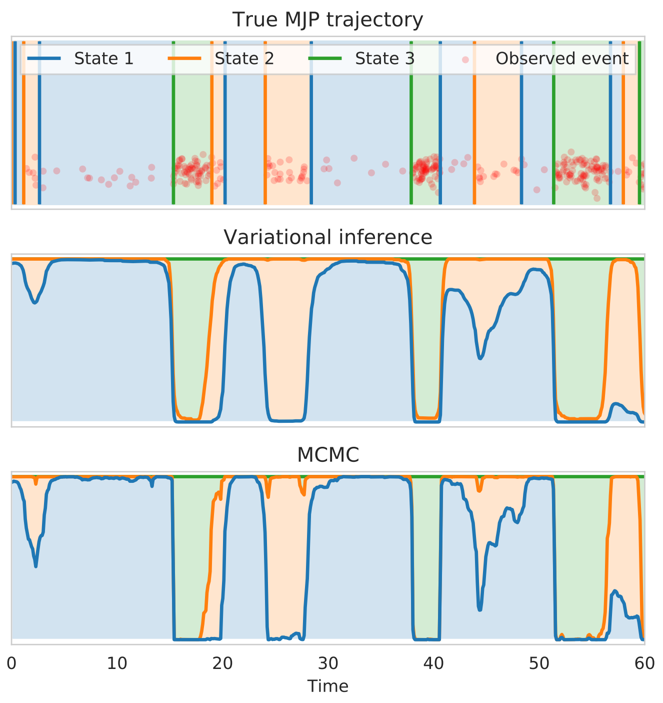


Figure 5.5: Posterior distributions over the latent trajectory of a Markov modulated Poisson process learned using our VI approach and the MCMC sampler from [149].

Sampling-based losses for TPPs arise naturally in applications such as reinforcement learning [108, 183] and adversarial training [198, 199, 202]. Existing works rely on the score function estimator when learning TPPs with such sampling-based losses. The reparametrization sampling method for TPPs that we introduced in this chapter provides an easy-to-implement low-variance alternative to the score function estimator.

Latent space models. TPPs governed by latent Markov dynamics have intractable likelihoods that require approximations [84, 197]. For MJPs, the state-of-the-art approach is the Gibbs sampler by Rao & Teh [149]. It allows to exactly sample from the posterior $p(\mathcal{T}, \mathcal{S}|\mathcal{X}, \boldsymbol{\psi})$, but is known to converge slowly if the parameters $\boldsymbol{\psi}$ are to be learned as well [205]. Existing variational inference approaches for MJPs can only learn a fixed time discretization [206] or estimate the marginal statistics of the posterior [141, 194]. In contrast, our method produces a full distribution over the jump times.

5.5 Experiments

5.5.1 Variational inference on simulated data

Setup. We apply our variational inference method from Section 5.3 for learning the posterior distribution over the latent trajectories of an Markov modulated Poisson process (MMPP). We parametrize the variational distribution $q(\mathcal{T})$ over the jump times

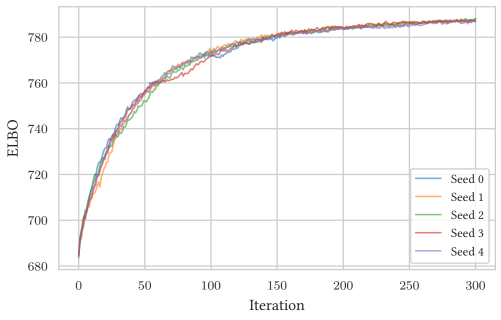


Figure 5.6: Convergence of our variational inference procedure with 5 different random seeds.

using the TriTPP model from Chapter 4. We simulate an MMPP with $K = 3$ latent states that correspond to different arrival rates of observed events $\mathcal{X}$. As a baseline, we use the state-of-the-art MCMC sampler by Rao & Teh [149]. In both cases we assume that the MMPP parameters ψ are known. See Appendix F.2

Results. Figure 5.5 shows the true latent MJP trajectory, as well as the marginal posterior probabilities learned by our method and the MCMC sampler of Rao & Teh. We can see that TriTPP accurately recovers the true posterior distribution over the trajectories. The three components that enable our new variational inference approach are our efficient parallel sampling algorithm for TriTPP (Section 4.2), reparametrization trick for TPPs (Section 5.2.1) and the differential relaxation (Section 5.2.2).

Convergence plots. Figure 5.6 additionally shows the convergence of the variational inference procedure by plotting the ELBO over the training iterations. As we see, we achieve stable optimization thanks to the differentiable relaxation of the loss.

Random initializations. In order to show that our results are not cherry-picked, we provide the plots of marginal posterior trajectories (similar to Figure 5.5) obtained with 3 different random seeds. Figure 5.7 shows that our results are consistent across the random seeds.

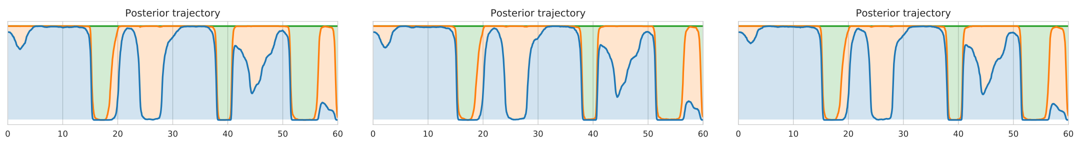


Figure 5.7: Marginal posterior trajectories obtained when using different random seeds.

5.5.2 Variational inference on real-world data

Setup. We apply our model to the server log data.[1] We simultaneously learn the posterior over the trajectories $(\mathcal{T}, \mathcal{S})$ as well as the model parameters $\boldsymbol{\psi} = \{\boldsymbol{\pi}, \boldsymbol{A}, \boldsymbol{\lambda}\}$ by

[1]`https://www.kaggle.com/shawon10/web-log-dataset`

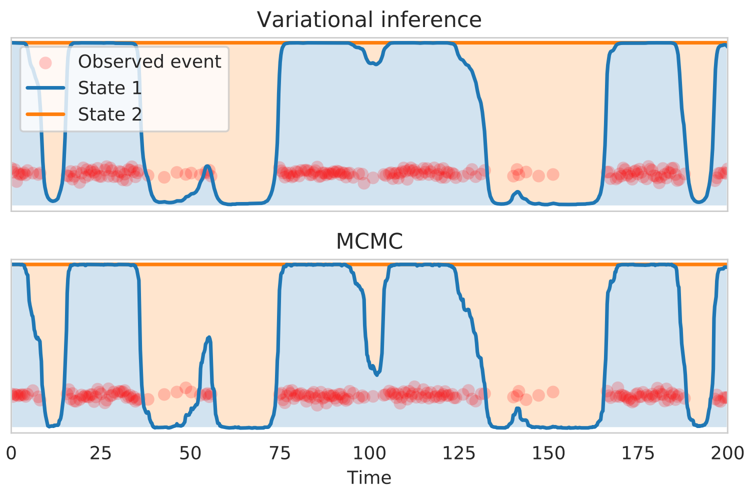


Figure 5.8: Segmentation of server data obtained using our VI approach and MCMC. In both cases, we estimate the posterior $p(\mathcal{T}, \mathcal{S}|\mathcal{X}, \boldsymbol{\psi})$ as well as the MMPP parameters $\boldsymbol{\psi}$.

solving the following optimization problem

$$\max_{\boldsymbol{\psi}} \max_{q(\mathcal{T},\mathcal{S})} \mathbb{E}_q[\log p(\mathcal{X}|\mathcal{T}, \mathcal{S}, \boldsymbol{\psi}) + \log p(\mathcal{T}, \mathcal{S}|\boldsymbol{\psi}) - \log q(\mathcal{T}, \mathcal{S})]. \tag{5.17}$$

Like before, we optimize a differentiable relaxation of this objective with gradient ascent. We compare our approach to the MCMC sampler of Rao & Teh as the baseline. For the MCMC sampler, we adopt an EM-like approach, where we alternate between closed-form parameter updates for $\boldsymbol{\psi}$ and simulating the posterior trajectories.

Results. Figure 5.8 shows the obtained posterior trajectories for the two approaches. Both models learn to segment the sequence into a high-event-rate and a low-event-rate states. This confirms that our variational inference approach is a viable alternative to the MCMC sampler.

5.6 Conclusions

We have shown how the reparametrization trick combined with a new differentiable relaxation allows us to train TPP models using sampling-based objective function. To demonstrate the utility of this framework, we developed an approximate posterior inference scheme for continuous-time discrete-state systems. Our generalization of the reparametrization trick can be combined with various TPP, and therefore lays the foundation for using TPPs as plug-and-play components of other machine learning models.

6 Anomaly detection

Temporal point processes (TPPs) provide a natural representation for transactions in financial systems, server logs, or user activity traces. Detecting anomalies in such data can provide immense industrial value. For example, abnormal entries in system logs may correspond to unnoticed server failures, atypical user activity in computer networks may correspond to intrusions, and irregular patterns in financial systems may correspond to fraud or shifts in the market structure.

Manual inspection of such event data is usually infeasible due to its sheer volume. At the same time, hand-crafted rules quickly become obsolete due to software updates or changing trends [81]. Ideally, we would like to have an adaptive system that can learn the normal behavior from the data, and automatically detect abnormal event sequences. Importantly, such a system should detect anomalies in a completely unsupervised way, as high-quality labels are usually hard to obtain.

Assuming "normal" data is available, we can formulate the problem of detecting anomalous event sequences as an instance of out-of-distribution (OoD) detection. Multiple recent works consider OoD detection for image data based on deep generative models [128, 154, 189]. However, none of these papers consider continuous-time event data. Neural TPPs that we discussed in earlier chapters define a generative model for such variable-length event sequences. Still, the literature on neural TPPs mostly focuses on prediction tasks, and the problem of anomaly detection has not been adequately addressed by existing works [170]. We aim to fill this gap in this chapter.

Our main contributions are the following:

- **Approach for anomaly detection with generative models.** We draw connections between OoD detection and GoF testing for TPPs (Section 6.1). By combining this insight with neural TPPs, we propose an approach for anomaly detection that shows high accuracy on synthetic and real-world event data.

- **A new test statistic for TPPs.** We highlight the limitations of popular GoF statistics for TPPs and propose the sum-of-squared-spacings statistic that addresses these shortcomings (Section 6.3). The proposed statistic can be applied to both unmarked and marked TPPs.

6.1 Anomaly detection and goodness-of-fit testing

Background. In this chapter we will use a slightly different notation compared to the rest of the thesis (see Appendices A and B for an overview). We use $\mathbb{P}$ to denote a TPP and the respective distribution over variable-length event sequences. We denote a TPP

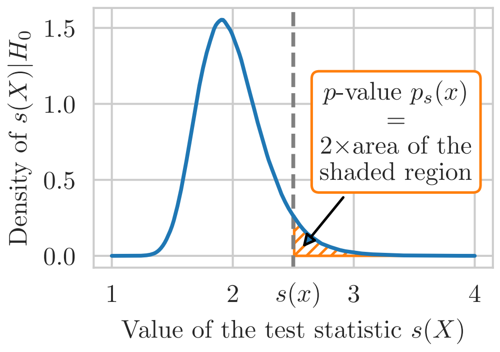


Figure 6.1: p-value is computed as the tail probability under the sampling distribution $s(X)|H_0$.

realization as $X = (t_1, \dots, t_N)$, where N, the number of events, is itself a random variable. Same as before, we can characterize a TPP using a conditional intensity function $\lambda^*(t) := \lambda(t|\mathcal{H}(t))$ that is equal to the rate of arrival of new events given the history $\mathcal{H}(t)$ consisting of past events (Equation 2.9). Equivalently, a TPP can be specified with the compensator $\Lambda^*(t) = \int_0^t \lambda^*(u)du$ (Equation 2.20).

Out-of-distribution (OoD) detection. We formulate the problem of detecting anomalous event sequences as an instance of OoD detection [109]. Namely, we assume that we are given a large set of training sequences $\mathcal{D}_{\text{train}} = \{X_1, \dots, X_M\}$ that were sampled i.i.d. from some *unknown* distribution $\mathbb{P}_{\text{data}}$ over a domain $\mathcal{X}$. At test time, we need to determine whether a new sequence X was also drawn from $\mathbb{P}_{\text{data}}$ (i.e., X is in-distribution or "normal") or from another distribution $\mathbb{Q} \neq \mathbb{P}_{\text{data}}$ (i.e., X is out-of-distribution or anomalous). We can phrase this problem as a null hypothesis test:

$$H_0\colon X \sim \mathbb{P}_{\text{data}} \qquad\qquad H_1\colon X \sim \mathbb{Q} \quad \text{for some} \quad \mathbb{Q} \neq \mathbb{P}_{\text{data}}. \tag{6.1}$$

To reiterate, here we consider the case where X is a variable-length event sequence and $\mathbb{P}_{\text{data}}$ is some unknown TPP. However, the rest of the discussion in Section 6.1 also applies to distributions over other data types, such as images.

Goodness-of-fit (GoF) testing. First, we observe that the problem of OoD detection is closely related to the problem of GoF testing [40]. We now outline the setup and approaches for GoF testing, and then describe how these can be applied to OoD detection. The goal of a GoF test to determine whether a random element X follows a *known* distribution $\mathbb{P}_{\text{model}}$[1]

$$H_0\colon X \sim \mathbb{P}_{\text{model}} \qquad\qquad H_1\colon X \sim \mathbb{Q} \quad \text{for some} \quad \mathbb{Q} \neq \mathbb{P}_{\text{model}}. \tag{6.2}$$

[1]We test a single realization X, as is common in TPP literature [22]. Note that this differs from works on *univariate* GoF testing that consider multiple realizations, i.e., $H_0\colon X_1, \dots, X_M \overset{\text{i.i.d.}}{\sim} \mathbb{P}_{\text{model}}$.

We can perform such a test by defining a test statistic $s(X)$, where $s\colon \mathcal{X} \to \mathbb{R}$ [62]. For this, we compute the (two-sided) p-value for an observed realization x of X as[2]

$$p_s(x) = 2 \times \min\{\Pr(s(X) \le s(x)|H_0), 1 - \Pr(s(X) \le s(x)|H_0)\}. \tag{6.3}$$

The factor 2 accounts for the fact that the test is two-sided. We reject the null hypothesis (i.e., conclude that X does not follow $\mathbb{P}_{\text{model}}$) if the p-value is below some predefined confidence level α. Note that computing the p-value requires evaluating the cumulative distribution function (CDF) of the sampling distribution, i.e., the distribution test statistic $s(X)$ under the null hypothesis H_0.

GoF testing vs. OoD detection. The two hypothesis tests (Equations 6.1 and 6.2) appear similar—in both cases the goal is to determine whether X follows a certain distribution $\mathbb{P}$ and no assumptions are made about the alternative $\mathbb{Q}$. This means that we can perform OoD detection using the procedure described above, that is, by defining a test statistic $s(X)$ and computing the respective p-value (Equation 6.3). However, in case of GoF testing (Equation 6.2), the distribution $\mathbb{P}_{\text{model}}$ is known. Therefore, we can analytically compute or approximate the CDF of $s(X)|X \sim \mathbb{P}_{\text{model}}$, and thus the p-value. In contrast, in an OoD detection hypothesis test (Equation 6.1), we make no assumptions about $\mathbb{P}_{\text{data}}$ and only have access to samples $\mathcal{D}_{\text{train}}$ that were drawn from this distribution. For this reason, we cannot compute the CDF of $s(X)|X \sim \mathbb{P}_{\text{data}}$ analytically. Instead, we can approximate the p-value using the empirical cumulative distribution function (eCDF) of the test statistic $s(X)$ on $\mathcal{D}_{\text{train}}$.

The above procedure can be seen as a generalization of many existing methods for unsupervised OoD detection. These approaches usually define the test statistic based on the log-likelihood (LL) of a generative model fitted to $\mathcal{D}_{\text{train}}$ [32, 154, 161]. However, as follows from our discussion above, there is no need to limit ourselves to LL-based statistics. For instance, we can define a test statistic for event sequences based on the rich literature on GoF testing for TPPs. We show in Section 6.5 that this often leads to more accurate anomaly detection compared to LL. Moreover, the difference between OoD detection and GoF testing is often overlooked. By drawing a clear distinction between the two, we can avoid some of the pitfalls encountered by other works [128], as we elaborate in Appendix G.1.

The anomaly detection framework we outlined above can be applied to any type of data—such as images or time series—but in this work we mostly focus on continuous-time event data. This means that our main goal is to find an appropriate test statistic for variable-length continuous-time event sequences. In Section 6.2, we take a look at existing GoF statistics for TPPs and analyze their limitations. Then in Section 6.3, we propose a new test statistic that addresses these shortcomings and describe in more detail how it can be used for OoD detection.

[2]In the rest of the paper, the difference between the random element X and its realization x is unimportant, so we denote both as X, as is usually done in the literature.

6.2 Review of existing GoF test statistics for TPPs

Here, we consider a GoF test (Equation 6.2), where the goal is to determine whether an event sequence $X = (t_1, \dots, t_N)$ was generated by a known TPP $\mathbb{P}_{\text{model}}$ with compensator Λ^*. We will return to the problem of OoD detection, where the data-generating distribution $\mathbb{P}_{\text{data}}$ is unknown, in Section 6.3.2.

Many popular GoF tests for TPPs are based on the random time change theorem (Theorem 1). The theorem states that a sequence $X = (t_1, \dots, t_N)$ is distributed according to a TPP with compensator Λ^* on the interval $[0, T]$ if and only if the sequence $Z = (\Lambda^*(t_1), \dots, \Lambda^*(t_N))$ is distributed according to the standard Poisson process (SPP) on $[0, \Lambda^*(T)]$. Intuitively, this result can be viewed as a TPP analogue of how the CDF of an arbitrary random variable over $\mathbb{R}$ transforms its realizations into samples from Uniform($[0, 1]$). Similarly, the compensator Λ^* converts a random event sequence X into a realization Z from the SPP.

Therefore, the problem of GoF testing for an arbitrary TPP reduces to testing whether the transformed sequence Z follows the SPP on $[0, \Lambda^*(T)]$. In other words, we can define a GoF statistic for a TPP with compensator Λ^* by (1) applying the compensator to X to obtain Z and (2) computing one of the existing GoF statistics for the SPP on the transformed sequence. This can also be generalized to marked TPPs (where events can belong to one of C classes) by simply concatenating the transformed sequences $Z^{(c)}$ for each event type $c \in \{1, \dots, C\}$ (see Appendix G.4 for details).

SPP, i.e., the Poisson process with constant intensity $\lambda^*(t) = 1$, is the most basic TPP one can conceive. However, as we will shortly see, existing GoF statistics even for this simple model have considerable shortcomings and can only detect a limited class of deviations from the SPP. More importantly, test statistics for general TPPs defined using the above recipe (Theorem 1) inherit the limitations of the SPP statistics.

For brevity, we denote the length of the transformed interval as $V = \Lambda^*(T)$ and the transformed arrival times as $Z = (v_1, \dots, v_N) = (\Lambda^*(t_1), \dots, \Lambda^*(t_N))$. One way to describe the generative process of an SPP is as follows [144]

$$N|V \sim \text{Poisson}(V) \qquad u_i|N, V \sim \text{Uniform}([0, V]) \quad \text{for } i = 1, \dots, N. \tag{6.4}$$

An SPP realization $Z = (v_1, \dots, v_N)$ is obtained by sorting the u_i's in increasing order. This is equivalent to defining the arrival time v_i as the i-th order statistic $u_{(i)}$. We can also represent Z by the inter-event times $(w_1, \dots, w_{N+1})$ where $w_i = v_i - v_{i-1}$, assuming $v_0 = 0$ and $v_{N+1} = V$.

Barnard [10] proposed a GoF test for the SPP based on the above description (Equation 6.4) and the Kolmogorov–Smirnov (KS) statistic. The main idea of this approach is to check whether the arrival times $v_1, \dots, v_N$ are distributed uniformly in the $[0, V]$ interval. For this, we compare $\hat{F}_{\text{arr}}$, the empirical CDF of the arrival times, with $F_{\text{arr}}(u) = u/V$, the CDF of the Uniform($[0, V]$) distribution. This can be done using the **KS statistic on the arrival times (KS arrival)**, defined as

$$\kappa_{\text{arr}}(Z) = \sqrt{N} \cdot \sup_{u \in [0, V]} |\hat{F}_{\text{arr}}(u) - F_{\text{arr}}(u)| \quad \text{where} \quad \hat{F}_{\text{arr}}(u) = \frac{1}{N} \sum_{i=1}^{N} \mathbb{1}(v_i \le u). \tag{6.5}$$

Another popular GoF test for the SPP is based on the fact that the inter-event times w_i are distributed according to the Exponential(1) distribution [36]. The test compares $\hat{F}_{\text{int}}$, the empirical CDF of the inter-event times, and $F_{\text{int}}(u) = 1 - \exp(-u)$, the CDF of the Exponential(1) distribution. This leads to the **KS statistic for the inter-event times (KS inter-event)**

$$\kappa_{\text{int}}(Z) = \sqrt{N} \cdot \sup_{u \in [0, \infty)} |\hat{F}_{\text{int}}(u) - F_{\text{int}}(u)| \quad \text{where} \quad \hat{F}_{\text{int}}(u) = \frac{1}{N+1} \sum_{i=1}^{N+1} \mathbb{1}(w_i \le u). \tag{6.6}$$

KS arrival and KS inter-event statistics are often presented as the go-to approach for testing the goodness-of-fit of the standard Poisson process [41]. Combining them with Theorem 1 leads to simple GoF tests for arbitrary TPPs that are widely used to this day [5, 66, 97, 108, 179].

Limitations of the KS statistics. The KS statistics $\kappa_{\text{arr}}(Z)$ and $\kappa_{\text{int}}(Z)$ are only able to differentiate the SPP from a narrow class of alternative processes. For example, KS arrival only checks if the arrival times v_i are distributed uniformly, conditioned on the event count N. But what if the observed N is itself extremely unlikely under the SPP (Equation 6.4)? KS inter-event can be similarly insensitive to the event count—removing all events $\frac{V}{2} < v_i \le V$ from an SPP realization Z will only result in just a single atypically large inter-event time w_i, which changes the value of $\kappa_{\text{int}}(Z)$ at most by $\frac{1}{N+1}$. We demonstrate these limitations of $\kappa_{\text{arr}}(Z)$ and $\kappa_{\text{int}}(Z)$ in our experiments in Section 6.5.1. Other failure modes of the KS statistics were described by Pillow [146]. Note that ad-hoc fixes to the KS statistics do not address these problems. For example, combining multiple tests performed separately for the event count and arrival times using Fisher's method [36, 63] consistently decreases the accuracy, as we show in Appendix G.7. In the next section, we introduce a different test statistic that aims to address these shortcomings.

6.3 Sum-of-squared-spacings (3S) statistic for TPPs

6.3.1 Goodness-of-fit testing with the 3S statistic

A good test statistic should capture multiple properties of the SPP at once: it should detect deviations w.r.t. both the event count N and the distribution of the arrival or inter-event times. Here, we propose to approach GoF testing with a **sum-of-squared-spacings (3S) statistic** that satisfies these desiderata,

$$\psi(Z) = \frac{1}{V} \sum_{i=1}^{N+1} w_i^2 = \frac{1}{V} \sum_{i=1}^{N+1} (v_i - v_{i-1})^2. \tag{6.7}$$

This statistic extends the sum-of-squared-spacings statistic proposed as a test of uniformity for fixed-length samples by Greenwood [75]. The important difference between our definition (Equation 6.7) and prior works [40] is that we, for the first time, consider the

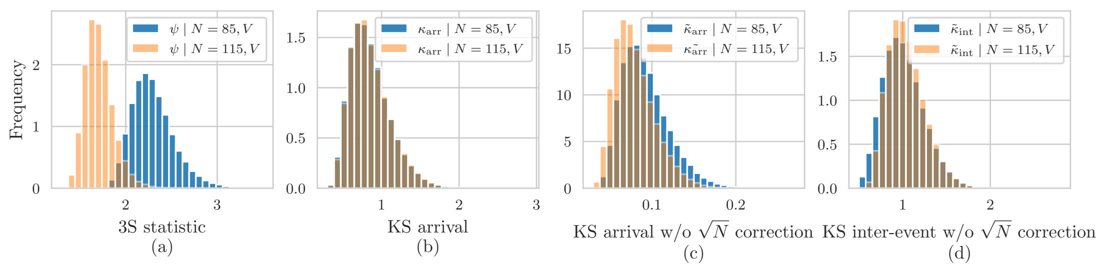


Figure 6.2: Distribution of different test statistics for the standard Poisson process on $[0, 100]$, conditioned on different event counts N. The 3S statistic allows us to differentiate between different values of N, while KS statistics are not sensitive to changes in N.

TPP setting, where the number of events N is random as well. For this reason, we use the normalizing constant $1/V$ instead of N/V^2 (see Appendix G.2 for details). As we will see, this helps capture abnormalities in the event count and results in more favorable asymptotic properties for the case of SPP.

Intuitively, for a fixed N, the statistic ψ is maximized if the spacings are extremely imbalanced, i.e., if one inter-event time w_i is close to V and the rest are close to zero. Conversely, ψ attains its minimum when the spacings are all equal, that is $w_i = \frac{V}{N+1}$ for all i.

In Figure 6.2a we visualize the distribution of $\psi | N, V$ for two different values of N. We see that the distribution of ψ depends strongly on N, therefore a GoF test involving ψ will detect if the event count N is atypical for the given SPP. This is in contrast to κ_{arr} and κ_{int}, the distributions of which, by design, are (asymptotically) invariant under N (Figure 6.2b). Even if one accounts for this effect, e.g., by removing the correction factor $\sqrt{N}$ in Equations 6.5 and 6.6, their distributions change only slightly compared to the sum of squared spacings (see Figures 6.2c and 6.2d). To analyze other properties of the statistic, we consider its moments under the null hypothesis.

Proposition 1. *Suppose the sequence Z is distributed according to the standard Poisson process on the interval $[0, V]$. Then the first two moments of the statistic $\psi := \psi(Z)$ are*

$$\mathbb{E}[\psi|V] = \frac{2}{V}(V + e^{-V} - 1) \quad \textit{and} \quad \operatorname{Var}[\psi|V] = \frac{4}{V^2}(2V - 7 + e^{-V}(2V^2 + 4V + 8 - e^{-V})).$$

The proof of Proposition 1 can be found in Appendix G.3. From Proposition 1 it follows that

$$\lim_{V\to\infty} \mathbb{E}[\psi|V] = 2 \qquad\qquad \lim_{V\to\infty} \operatorname{Var}[\psi|V] = 0. \tag{6.8}$$

This leads to a natural notion of *typicality* in the sense of Nalisnick et al. [128] and Wang et al. [189] for the standard Poisson process. We can define the typical set of the SPP as the set of variable-length sequences Z on the interval $[0, V]$ that satisfy $|\psi(Z) - 2| \leq \epsilon$ for some small $\epsilon > 0$. It follows from Equation 6.8 and Chebyshev's inequality that for large enough V, the SPP realizations will fall into the typical set with

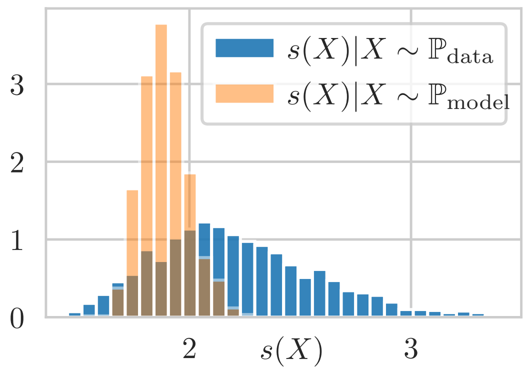


Figure 6.3: Sampling distribution for the OoD test (blue) and the GoF test (orange). While the same statistic $s(X)$ is used in both cases, the p-values are computed differently depending on which test we perform.

high probability. Therefore, at least for large V, we should be able to detect sequences that are not distributed according the SPP based on the statistic ψ.

Summary. To test the GoF of a TPP with a known compensator Λ^* for an event sequence $X = (t_1, \dots, t_N)$, we first obtain the transformed sequence $Z = (\Lambda^*(t_1), \dots, \Lambda^*(t_N))$ and compute the statistic $\psi(Z)$ according to Equation 6.7. Since the CDF of the statistic under H_0 cannot be computed analytically, we approximate it using samples drawn from $\mathbb{P}_{\text{model}}$. That is, we draw realizations $\mathcal{D}_{\text{model}} = \{X_1, \dots, X_M\}$ from the TPP (e.g., using the inversion method [153]) and compute the p-value for X (Equation 6.3) using the eCDF of the statistic on $\mathcal{D}_{\text{model}}$ [131].

6.3.2 Out-of-distribution detection with the 3S statistic

We now return to the original problem of OoD detection in TPPs, where we have access to a set of in-distribution sequences $\mathcal{D}_{\text{train}}$ and do not know the data-generating process $\mathbb{P}_{\text{data}}$.

Our idea is to perform the OoD detection hypothesis test (Equation 6.1) using the sum-of-squared-spacings test statistic that we introduced in the previous section. However, since the data-generating TPP $\mathbb{P}_{\text{data}}$ is unknown, we do not know the corresponding compensator that is necessary to compute the statistic. Instead, we can fit a neural TPP model $\mathbb{P}_{\text{model}}$ [52] to the sequences in $\mathcal{D}_{\text{train}}$ and use the compensator Λ^* of the learned model to compute the statistic $s(X)$.[3] High flexibility of neural TPPs allows these models to more accurately approximate the true compensator. Having defined the statistic, we can approximate its distribution under H_0 (i.e., assuming $X \sim \mathbb{P}_{\text{data}}$) by the eCDF of the statistic on $\mathcal{D}_{\text{train}}$. We use this eCDF to compute the p-values for our OoD detection hypothesis test and thus detect anomalous sequences. We provide the pseudocode description of our OoD detection method in Appendix G.4.

[3]We can replace the 3S statistic on the transformed sequence Z with any other statistic for the SPP, such as KS arrival. In Sections 6.5.2 and 6.5.3, we compare different statistics constructed this way.

We highlight that an OoD detection procedure like the one above is *not* equivalent to a GoF test for the learned generative model $\mathbb{P}_{\text{model}}$, as suggested by earlier works [128]. While we use the compensator of the learned model to define the test statistic $s(X)$, we compute the p-value for the OoD detection test based on $s(X)|X \sim \mathbb{P}_{\text{data}}$. This is different from the distribution $s(X)|X \sim \mathbb{P}_{\text{model}}$ used in a GoF test, since in general $\mathbb{P}_{\text{model}} \neq \mathbb{P}_{\text{data}}$. Therefore, even if the distribution of a test statistic under the GoF test can be approximated analytically (as, e.g., for the KS statistic [117]), we have to use the eCDF of the statistic on $\mathcal{D}_{\text{train}}$ for the OoD detection test. Figure 6.3 visualizes this difference. Here, we fit a TPP model on the in-distribution sequences from the STEAD dataset (Section 6.5.3) and plot the empirical distribution of the respective statistic $s(X)$ on $\mathcal{D}_{\text{train}}$ (corresponds to $s(X)|X \sim \mathbb{P}_{\text{data}}$) and on model samples $\mathcal{D}_{\text{model}}$ (corresponds to $s(X)|X \sim \mathbb{P}_{\text{model}}$).

6.4 Related work

Unsupervised OoD detection. OoD detection approaches based on deep generative models (similar to our approach in Section 6.3.2) have received a lot of attention in the literature. However, there are several important differences between our method and prior works. First, most existing approaches perform OoD detection based on the log-likelihood (LL) of the model or some derived statistic [32, 125, 128, 154, 161]. We observe that LL can be replaced by any other test statistic, e.g., taken from the GoF testing literature, which often leads to more accurate anomaly detection (Section 6.5). Second, unlike prior works, we draw a clear distinction between OoD detection and GoF testing. While this difference may seem obvious in hindsight, it is not acknowledged by the existing works, which may lead to complications (see Appendix G.1). Also, our formulation of the OoD detection problem in Section 6.1 provides an intuitive explanation to the phenomenon of "typicality" [128, 189]. The $(\epsilon, 1)$-typical set of a distribution $\mathbb{P}$ simply corresponds to the acceptance region of the respective hypothesis test with confidence level ϵ (Equation 6.1). Finally, most existing papers study OoD detection for image data and none consider variable-length event sequences, which is the focus of our work.

Our OoD detection procedure is also related to the *rarity* anomaly score [60, 93]. The rarity score can be interpreted as the negative logarithm of a one-sided p-value (Equation 6.3) of a GoF test that uses the log-likelihood of some *known* model as the test statistic. In contrast, we consider a broader class of statistics and learn the model from the data.

Anomaly detection for TPPs. OoD detection, as described in Section 6.1, is not the only way to formalize anomaly detection for TPPs. For example, [137] developed a distance-based approach for Poisson processes. Recently, [214] proposed to detect anomalous event sequences with an adversarially-trained model. Unlike these two methods, our approach can be combined with any TPP model without altering the training procedure. [114] studied anomalous *event* detection with TPPs, while we are concerned with entire event sequences.

GoF tests for TPPs. Existing GoF tests for the SPP usually check if the arrival times are distributed uniformly, using, e.g., the KS [107] or chi-squared statistic [35]. Our 3S statistic favorably compares to these approaches thanks to its dependence on the event count N, as we explain in Section 6.3 and show experimentally in Section 6.5.1. Methods combining the random time change theorem with a GoF test for the SPP (usually, the KS test) have been used at least since Ogata [134], and are especially popular in neuroscience [22, 66, 179]. However, these approaches inherit the limitations of the underlying KS statistic. Replacing the KS score with the 3S statistic consistently leads to a better separation between different TPP distributions (Section 6.5).

Gerhard and Wulfram [65] discussed several GoF tests for discrete-time TPPs, while we deal with continuous time. Yang et al. [203] proposed a GoF test for point processes based on Stein's identity, which is related to a more general class of kernel-based GoF tests [34, 113]. Their approach is not suitable for neural TPPs, where the Papangelou intensity cannot be computed analytically. A recent work by [191] designed a GoF test for self-exciting processes under model misspecification. In contrast to these approaches, our proposed GoF test from Section 6.3.1 can be applied to any TPP with a known compensator.

Sum-of-squared-spacings statistic. A similar statistic was first used by Greenwood [75] for testing whether a fixed number of points are distributed uniformly in an interval. Several follow-up works studied the limiting distribution of the statistic (conditioned on N) as $N \to \infty$ [83, 148, 174]. Our proposed statistic (Equation 6.7) is not invariant w.r.t. N and, therefore, is better suited for testing TPPs. We discuss other related statistics in Appendix G.2.

6.5 Experiments

Our experimental evaluation covers two main topics. In Section 6.5.1, we compare the proposed 3S statistic with existing GoF statistics for the SPP. Then in Sections 6.5.2 and 6.5.3, we evaluate our OoD detection approach on simulated and real-world data, respectively. The experiments were run on a machine with a 1080Ti GPU. Details on the setup and datasets construction are provided in Appendix G.5 & G.6.

6.5.1 Standard Poisson process

In Section 6.2 we mentioned several failure modes of existing GoF statistics for the SPP. Then, in Section 6.3.1 we introduced the 3S statistic that was supposed to address these limitations. Hence, the goal of this section is to compare the proposed statistic with the existing ones in the task of GoF testing for the SPP. We consider four test statistics: (1) KS statistic on arrival times (Equation 6.5), (2) KS statistic on inter-event times (Equation 6.6), (3) chi-squared statistic on the arrival times [35, 179], and (4) the proposed 3S statistic (Equation 6.7).

To quantitatively compare the discriminative power of different statistics, we adopt an evaluation strategy similar to [65] and [203]. First, we generate a set $\mathcal{D}_{\text{model}}$ consisting of 1000 SPP realizations. We use $\mathcal{D}_{\text{model}}$ to compute the empirical distribution function of

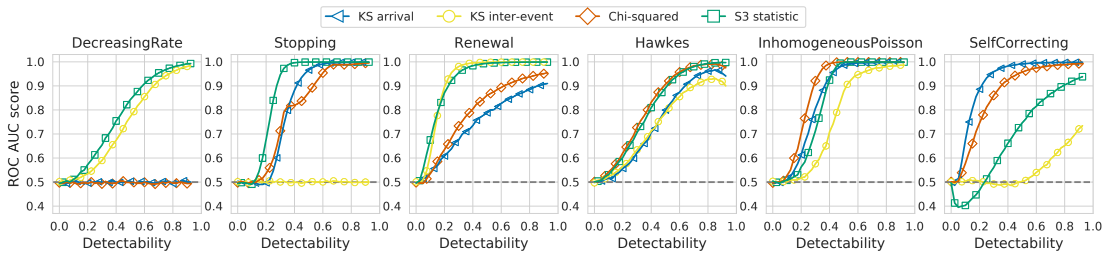


Figure 6.4: GoF testing for the standard Poisson process using different test statistics, measured with ROC AUC (higher is better). See Section 6.5.1 for the description of the experimental setup.

each statistic $s(Z)$ under H_0. Then, we define two test sets: $\mathcal{D}_{\text{test}}^{\text{ID}}$ (consisting of samples from $\mathbb{P}_{\text{model}}$, the SPP) and $\mathcal{D}_{\text{test}}^{\text{OOD}}$ (consisting of samples from $\mathbb{Q}$, another TPP), each with 1000 sequences. Importantly, in this and following experiments, the training and test sets are always disjoint.

We follow the GoF testing procedure described at the end of Section 6.3.1, which corresponds to the hypothesis test in Equation 6.2. That is, we compute the p-value (Equation 6.3) for each sequence in the test sets using the eCDF of $s(Z)$ on $\mathcal{D}_{\text{model}}$. A good test statistic $s(Z)$ should assign lower p-values to the OoD sequences from $\mathcal{D}_{\text{test}}^{\text{OOD}}$ than to ID sequences from $\mathcal{D}_{\text{test}}^{\text{ID}}$, allowing us to discriminate between samples from $\mathbb{Q}$ and $\mathbb{P}_{\text{model}}$. We quantify how well a given statistic separates the two distributions by computing the area under the ROC curve (ROC AUC). This effectively averages the performance of a statistic for the GoF hypothesis test over different significance levels α.

Datasets. We consider six choices for the distribution $\mathbb{Q}$:

- Rate, a homogeneous Poisson process with intensity $\mu < 1$;
- Stopping, where events stop after some time $t_{\text{stop}} \in [0, V]$;
- Renewal, where inter-event times are drawn i.i.d. from the Gamma distribution;
- Hawkes, where events are more clustered compared to the SPP;
- Inhomogeneous, a Poisson process with non-constant intensity $\lambda(t) = \beta \sin(\omega t)$;
- SelfCorrecting, where events are more evenly spaced compared to the SPP.

For cases the last 4 cases, the expected number of events is the same as for the SPP.

For each choice of $\mathbb{Q}$ we define a *detectability* parameter $\delta \in [0, 1]$, where higher δ corresponds to TPPs that are increasingly dissimilar to the SPP. That is, setting $\delta = 0$ corresponds to a distribution $\mathbb{Q}$ that is exactly equal to the SPP, and $\delta = 1$ corresponds to a distribution that deviates significantly from the SPP. For example, for a Hawkes with conditional intensity $\lambda^*(t) = \mu + \beta \sum_{t_j < t} \exp(-(t - t_j))$, the detectability value of $\delta = 0$ corresponds to $\mu = 1$ and $\beta = 0$ (i.e., $\lambda^*(t) = 1$) making $\mathbb{Q}$ indistinguishable from $\mathbb{P}$. The value of $\delta = 0.5$ corresponds to $\mu = 0.5$ and $\beta = 0.5$, which preserves the expected number of events N but makes the arrival times t_i "burstier." We describe how the parameters of each distribution $\mathbb{Q}$ are defined based on δ in Appendix G.5. Note that, in general, the ROC AUC scores are not guaranteed to monotonically increase as the detectability δ is increased.

Results. In Figure 6.4, we present AUC scores for different statistics as δ is varied. As expected, KS arrival accurately identifies sequences that come from $\mathbb{Q}$ where the absolute time of events are non-uniform (as in INHOMOGENEOUS). Similarly, KS inter-event is good at detecting deviations in the distribution of inter-event times, as in RENEWAL. The performance of the chi-squared statistic is similar to that of KS arrival. Nevertheless, the above statistics fail when the expected number of events, N, changes substantially—as in KS arrival and chi-squared on RATE, and KS inter-event on STOPPING. These failure modes match our discussion from Section 6.2.

In contrast, the 3S statistic stands out as the most consistent test (best or close-to-best performance in 5 out of 6 cases) and does not completely fail in any of the scenarios. The relatively weaker performance on SELFCORRECTING implies that the 3S statistic is less sensitive to superuniform spacings [40] than to imbalanced spacings. The results show that the 3S statistic is able to detect deviations w.r.t. both the event count N (RATE and STOPPING), as well as the distributions of the inter-event times w_i (RENEWAL) or the arrival times v_i (HAWKES and INHOMOGENEOUS)—something that other GoF statistics for the SPP cannot provide.

6.5.2 Detecting anomalies in simulated data

In this section, we test the OoD detection approach discussed in Section 6.3.2, i.e., we perform anomaly detection for a TPP with an *unknown* compensator. This corresponds to the hypothesis test in Equation 6.1. We use the training set $\mathcal{D}_{\text{train}}$ to fit an RNN-based neural TPP model [167] via maximum likelihood estimation (see Appendix G.6 for details). Then, we define test statistics for the general TPP as follows. We apply the compensator Λ^* of the learned model to each event sequence X and compute the four statistics for the SPP from Section 6.5.1 on the transformed sequence $Z = \Lambda^*(X)$. We highlight that these methods are not "baselines" in the usual sense—the idea of combining a GoF statistic with a learned TPP model to detect anomalous event sequences is itself novel and has not been explored by earlier works. The rest of the setup is similar to Section 6.5.1. We use $\mathcal{D}_{\text{train}}$ to compute the eCDF of each statistic under H_0, and then compute the ROC AUC scores on the p-values. In addition to the four statistics discussed before, we consider a two-sided test on the log-likelihood $\log q(X)$ of the learned generative model, which corresponds to the approach by Nalisnick et al. [128].

Datasets. Like before, we define a detectability parameter δ for each scenario that determines how dissimilar ID and OoD sequences are. SERVER-STOP, SERVER-OVERLOAD and LATENCY are inspired by applications in DevOps, such as detecting anomalies in server logs.

- SERVER-OVERLOAD and SERVER-STOP contain data generated by a multivariate Hawkes process with 3 marks, e.g., modeling network traffic among 3 hosts. In

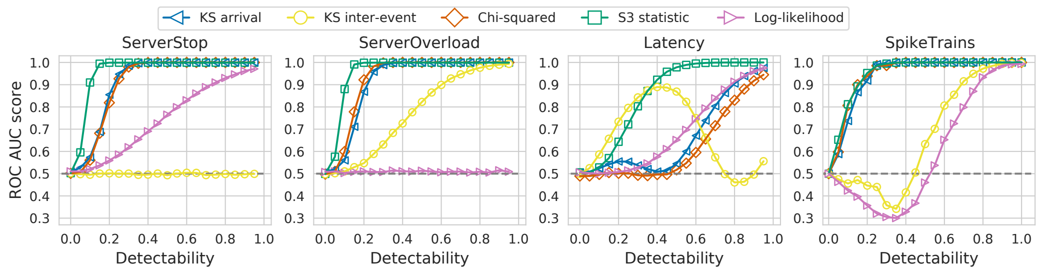


Figure 6.5: OoD detection on simulated data using different test statistics, measured with with ROC AUC (higher is better). See Section 6.5.2 for the description of the experimental setup.

OoD sequences, we change the influence matrix to simulate scenarios where a host goes offline (Server-Stop), and where a host goes down and the traffic is routed to a different host (Server-Overload). Higher δ implies that the change in the influence matrix happens earlier.

- Latency contains events of two types, sampled as follows. The first mark, the "trigger," is sampled from a homogeneous Poisson process with rate $\mu = 3$. The arrival times of the second mark, the "response," are obtained by shifting the times of the first mark by an offset sampled i.i.d. from Normal$(\mu = 1, \sigma = 0.1)$. In OoD sequences, the delay is increased by an amount proportional to δ, which emulates an increased latency in the system.

- SpikeTrains [175] contains sequences of firing times of 50 neurons, each represented by a distinct mark. We generate OoD sequences by shuffling the indices of k neurons (e.g., switching marks 1 and 2), where higher detectability δ implies more switches k. Here we study how different statistics behave for TPPs with a large number of marks.

Results are shown in Figure 6.5. The 3S statistic demonstrates excellent performance in all four scenarios, followed by KS arrival and chi-squared. In case of Server-Stop and Server-Overload, the 3S statistic allows us to perfectly detect the anomalies even when only 5% of the time interval are affected by the change in the influence structure. KS inter-event and log-likelihood statistics completely fail on Server-Stop and Server-Overload, respectively. These two statistics also struggle to discriminate OoD sequences in Latency and SpikeTrains scenarios. The non-monotone behavior of the ROC AUC scores for some statistics (as the δ increases) indicates that these statistics are poorly suited for the respective scenarios.

6.5.3 Detecting anomalies in real-world data

Finally, we apply our methods to detect anomalies in two real-world event sequence datasets. We keep the setup (e.g., configuration of the neural TPP model) identical to Section 6.5.2.

Table 6.1: ROC AUC scores for OoD detection on real-world datasets (mean & standard error are computed over 5 runs). Best result in **bold**, results within 2 pp. of the best $\underline{\text{underlined}}$.

	KS arrival	KS inter-event	Chi-squared	Log-likelihood	3S statistic
Logs — Packet corruption (1%)	57.4 ± 1.7	62.1 ± 0.9	66.6 ± 1.8	75.9 ± 0.1	**95.5** ± 0.3
Logs — Packet corruption (10%)	59.2 ± 2.3	$\underline{97.8}$ ± 0.6	59.1 ± 2.3	$\underline{99.0}$ ± 0.0	**99.4** ± 0.1
Logs — Packet duplication (1%)	81.1 ± 5.2	82.8 ± 5.0	74.6 ± 6.5	88.1 ± 0.1	**90.9** ± 0.3
Logs — Packet delay (frontend)	95.6 ± 1.2	$\underline{98.9}$ ± 0.4	**99.3** ± 0.1	90.9 ± 0.0	$\underline{97.6}$ ± 0.1
Logs — Packet delay (all services)	**99.8** ± 0.0	94.7 ± 1.1	**99.8** ± 0.0	96.1 ± 0.0	$\underline{99.6}$ ± 0.1
STEAD — Anchorage, AK	59.6 ± 0.2	79.7 ± 0.1	67.4 ± 0.2	$\underline{88.0}$ ± 0.1	**88.3** ± 0.6
STEAD — Aleutian Islands, AK	53.8 ± 0.5	88.8 ± 0.3	62.2 ± 0.9	$\underline{97.0}$ ± 0.0	**99.8** ± 0.0
STEAD — Helmet, CA	59.1 ± 0.9	**98.7** ± 0.0	70.0 ± 0.6	$\underline{96.9}$ ± 0.0	92.6 ± 0.3

Logs: We generate server logs using Sock Shop microservices [190] and represent them as marked event sequences. Sock Shop is a standard testbed for research in microservice applications [2] and contains a web application that runs on several containerized services. We generate OoD sequences by injecting various failures (e.g., packet corruption, increased latency) among these microservices using a chaos testing tool Pumba [104]. We split one large server log into 30-second subintervals, that are then partitioned into train and test sets.

STEAD (Stanford Earthquake Dataset) [126] includes detailed seismic measurements on over 1 million earthquakes. We construct four subsets, each containing 72-hour subintervals in a period of five years within a 350km radius of a fixed geographical location. We treat sequences corresponding the San Mateo, CA region as in-distribution data, and the remaining 3 regions (Anchorage, AK, Aleutian Islands, AK and Helmet, CA) as OoD data.

Results. Table 6.1 shows the ROC AUC scores for all scenarios. KS arrival and chi-squared achieve surprisingly low scores in 6 out of 8 scenarios, even though these two methods showed strong results on simulated data in Sections 6.5.1 and 6.5.2. In contrast, KS inter-event and log-likelihood perform better here than in previous experiments, but still produce poor results on Packet corruption. The 3S statistic is the only method that consistently shows high ROC AUC scores across all scenarios. Moreover, we observe that for *marked* sequences (Logs and all datasets in Section 6.5.2), the 3S statistic leads to more accurate detection compared to the log-likelihood statistic in 9 out of 9 cases.

6.6 Conclusions

Limitations. Our approach assumes that the sequences in $\mathcal{D}_{\text{train}}$ were drawn i.i.d. from the true data-generating distribution $\mathbb{P}_{\text{data}}$ (Section 6.1). This assumption can be violated in two ways: some of the training sequences might be anomalous or there might exist dependencies between them. We have considered the latter case in our experiments on SpikeTrains and Logs datasets, where despite the non-i.i.d. nature of the data our method was able to accurately detect anomalies. However, there might exist scenarios where the violation of the assumptions significantly degrades the performance.

No single test statistic can be "optimal" for either OoD detection or GoF testing, since we make no assumptions about the alternative distribution $\mathbb{Q}$ (Section 6.1). We empirically showed that the proposed 3S statistic compares favorably to other choices over a range of datasets and applications domains. Still, for any *fixed* pair of distributions $\mathbb{P}$ and $\mathbb{Q}$, one can always find a statistic that will have equal or higher power s.t. the same false positive rate [130]. Hence, it won't be surprising to find cases where our (or any other chosen a priori) statistic is inferior.

Broader impact. Continuous-time variable-length event sequences provide a natural representation for data such as electronic health records [56], server logs [81] and user activity traces [214]. The ability to perform unsupervised anomaly detection in such data can enable practitioners to find at-risk patients, reduce DevOps costs, and automatically detect security breaches—all of which are important tasks in the respective fields. One of the risks when applying an anomaly detection method in practice is that the statistical anomalies found by the method will not be relevant for the use case. For example, when looking for health insurance fraud, the method might instead flag legitimate patients who underwent atypically many procedures as "suspicious" and freeze their accounts. To avoid such situations, automated decisions systems should be deployed with care, especially in sensitive domains like healthcare.

Conclusion. We have presented an approach for OoD detection for temporal point processes based on goodness-of-fit testing. At the core of our approach lies a new GoF test for standard Poisson processes based on the 3S statistic. Our method applies to a wide class of TPPs and is extremely easy to implement. We empirically showed that the proposed approach leads to better OoD detection accuracy compared to both popular GoF statistics for TPPs (Kolmogorov–Smirnov, chi-squared) and approaches commonly used in OoD detection literature (model log-likelihood). While our analysis focuses on TPPs, we believe our discussion on similarities and distinctions between GoF testing and OoD detection offers insights to the broader machine learning community.

Part IV

Conclusion

7 Conclusion

In this thesis we discussed various aspects of design and application of temporal point processes for modeling continuous-time event data. In Part II, we presented two families of neural TPP models based on recurrent neural networks and triangular maps. In addition to being flexible, these models permit reparametrization sampling and can be trained efficiently via maximum likelihood. These properties open new applications for TPPs, as we showed in Part III. More specifically, we demonstrated how TPPs can be used for anomaly detection and trained using sampling-based losses.

We conclude this thesis with a retrospective, where we discuss subsequent works in the field of TPPs and show how our contributions fit into the broader research context. Finally, we list open research questions and possible directions for future work.

7.1 Retrospective

7.1.1 Neural TPP architectures

The LogNormMix model that we introduced in Chapter 3 follows the general encoder-decoder architecture for neural TPPs: The encoder embeds the event history $\mathcal{H}(t_i)$ into a summary vector $\boldsymbol{h}_i$, and the decoder uses $\boldsymbol{h}_i$ to model the distribution $p_i^*(t_i)$ of the next event. The novelty of our approach lies in a new decoder parametrization that enables efficient training and sampling, while the encoder is based on a recurrent neural network, similar to earlier works [52, 139].

Transformer. Several subsequent works [120, 208, 215, 217] suggested replacing the RNN encoder with a transformer [185]. Sharma et al. [166] have shown that our LogNormMix model can similarly be combined with a transformer encoder. The main advantage of the transformer compared to the RNN is its ability to capture long-range dependencies between events thanks to the self-attention mechanism. This, however, comes at increased computational cost — time and space complexity of evaluating the log-likelihood for a sequence with N events scale as $O(N^2)$ for a transformer. The RNN encoder is more efficient with its $O(N)$ scaling. This means that transformer-based neural TPPs cannot be trained on very long sequences (more than 10^3–10^4 events) but can achieve superior predictive results compared to RNN-based models [208].

Neural ordinary differential equations (neural ODEs). Another line of research explored TPP architectures based on neural ODEs [95, 159]. Such approaches define a state $\boldsymbol{h}(t)$ that evolves in continuous time according to a neural ODE [30]. The intensity $\lambda^*(t)$ at each time $t \in [0, T]$ is then defined directly as a function of the state $\lambda^*(t) = g(\boldsymbol{h}(t))$. This is different from encoder-decoder architectures that we discussed before, where the history embedding $\boldsymbol{h}_i$ is only updated after observed events. ODE-based models are, in theory, more flexible since they do not assume a parametric form for the conditional intensity function. However, training and prediction in such models rely on numerical integration, and therefore are slower and less accurate than in encoder-decoder TPPs. A recent work by Bilos et al. [13] showed a way to overcome this limitation by directly parametrizing the ODE solution. Their model, known as Neural Flow, can also be combined with LogNormMix in an encoder-decoder architecture.

Summary. Different families of neural TPP architectures (RNN-, transformer- and ODE-based) provide different trade-offs between efficiency, expressiveness and ability to capture long-range dependencies. Unfortunately, a thorough and fair comparison between different methods is difficult due to lack of standardized implementations and benchmarks. We will discuss this aspect in more detail in Section 7.2.

7.1.2 Reparametrization sampling for TPPs

In Part II, we presented LogNormMix and TriTPP — two neural TPP models that permit reparametrization sampling, and therefore can be trained using sampling-based losses. Later, Chen et al. [28] derived a similar reparametrization sampling method for TPP models based on neural ODEs. As in case of TriTPP, the approach of Chen et al. is based on the inverse transform algorithm (Algorithm 3).

Kajino [96] took a different approach and derived a differentiable relaxation of the thinning algorithm (Algorithm 5) for the spike-response model [68]. This approach is motivated by variational inference in spiking neural networks [156]. Their method uses the Gumbel-softmax trick [92, 116] to deal with non-differentiability of the loss function.

These different approaches for optimizing sampling-based losses all rely on specific properties of the underlying TPP model. For example, the spike-response model from [96] has a straightforward upper bound on the intensity, which is a prerequisite for applying the thinning algorithm. On the other hand, reparametrization trick for LogNormMix relies on our ability to differentiate through samples from the mixture distribution.

In conclusion, the availability of reparametrization sampling methods opens new applications for TPP models that go beyond the standard prediction tasks.

7.1.3 Applications

LogNormMix model (Chapter 3) can be trained and sampled from efficiently, which led to the adoption of this model in several follow-up works. For example, it has been used to simulate communication between computers, with applications in cybersecurity [122] and distributed system planning [173]. Gupta et al. [78] use LogNormMix to impute missing observations in partially-observed event sequences. Two other works extend our model to cluster event sequences, which is used to detect coordinated malicious accounts on social media [166, 211].

Anomaly detection. In Chapter 6, we presented an approach for detecting anomalous event sequences with neural TPP models. We consider *sequence-level* anomalies, where an entire event sequence can be either normal or anomalous. Such formulation can be used to find suspicious activity traces corresponding to fraudulent behavior in cybersecurity, or to find periods of abnormal activity in server logs. However, there exist other ways to formulate the anomaly detection task for TPPs. For instance, a concurrent work by Liu and Hauskrecht [114] aims to detect individual events that might be abnormal.

More broadly, the problem of unsupervised out-of-distribution detection with generative models has received a lot of attention in the machine learning community [27, 128, 129]. Similar to our approach in Chapter 6, Bergamin et al. [11] observed the close connection between the goodness-of-fit testing problem and out-of-distribution detection. They repurpose several general goodness-of-fit statistics based on the likelihood function, while we considered statistics for TPP models based on the random time change theorem.

Marked event sequences. In this thesis, we primarily focus on modeling unmarked event sequences, where events are represented only by their arrival time. An important line of ongoing research deals with marked TPPs, where each event contains additional metadata like type or location. For instance, *structure discovery* [209, 210] aims to detect causal relationships between different event types using TPPs with categorical marks. *Spatio-temporal* point processes [29, 213] predict event locations in addition to their times, which is useful in domains such as earthquake forecasting. Our LogNormMix model present can also be extended to the marked setting, as outlined in Section 3.2.4 and demonstrated by subsequent works [110]. Extending the TriTPP model (Chapter 4) to the marked setting, however, is more challenging and remains an important direction for future work.

7.2 Open questions and future work

Temporal point processes are now firmly established in the machine learning community, and an ever-increasing number of papers is published on this topic every year. Many of the works (including ones that constitute this thesis) focus on developing new neural TPP architectures [52, 119, 139, 167, 208, 215, 217]. We believe that some of the potentially most impactful directions for future work involve reconsidering how we evaluate and apply neural TPP models. We broadly divide these into the following categories.

Evaluation metrics. The two most commonly used approaches for evaluating the predictive performance of neural TPPs are based on negative log-likelihood (NLL) and single-event prediction (e.g., predicting time or type of the next event). Both of these have their disadvantages: NLL can be misleading and does not guarantee good sample quality [180], while single-event-based metrics provide limited insight into TPPs as generative models for entire event sequences [170, Section 7.2].

To guide our search for better TPP models we need metrics that are more closely aligned with the practical application of these models. For example, we can look into the field of seismology, where TPPs are used to forecast aftershock sequences [162]. Tools from probabilistic forecasting [69, 70] could be adopted to measure the quality of long-term forecasts generated by TPPs. Such metrics would both be relevant to practitioners and properly evaluate TPPs as generative models for entire sequences — two properties that are not satisfied by NLL and single-event-based metrics.

Datasets and benchmarks. Progress in many other subfields of machine learning has been driven by the availability of large high-quality datasets [47, 48, 87, 111]. Unfortunately, no such collections of event data are available for training and evaluating TPP models. Many event sequence datasets used in the literature are not motivated by any particular real-world application of TPPs. Therefore, it is unclear whether models that perform well on these datasets will generalize to real-world tasks. Enguehard et al. [56] also point out that several popular TPP datasets can be perfectly modeled by a simple history-independent baseline, so they are likely poorly suited for evaluating neural TPPs.

Another related issue is the lack of reference implementations of neural TPP models. A typical neural TPP implementation requires many preprocessing, hyperparameter and architectural choices. All of these vary greatly across different implementations, which makes it hard to pinpoint the source of improved performance. To conclude, developing standardized open-source neural TPP libraries as well as collecting high-quality event sequence datasets are both crucial for the progress of TPP research.

Applications. Many earlier developments in the field of TPPs were motivated by applications in scientific disciplines like seismology [134, 135] and neuroscience [46, 68]. Conventional TPP models like Hawkes and Neymann–Scott processes are still widely used by researchers in these domains [136, 195]. However, most of the recent work on neural TPPs stays limited to the machine learning community and has not propagated back to the practitioners.

Adapting neural TPPs to the traditional domains like seismology and neuroscience is important for several reasons. On the one hand, such work has practical significance for the respective fields. On the other hand, these application areas come with their own challenges, such as need for scalable and interpretable models. Addressing these challenges will in turn require innovation on the machine learning side of neural TPPs. There also exist other domains like information security, DevOps, demand forecasting and process mining that are full with potential applications for TPP models.

Bibliography

[1] Ryan Prescott Adams, Iain Murray, and David J. C. MacKay. Tractable nonparametric Bayesian inference in Poisson processes with Gaussian process intensities. In *International Conference on Machine Learning*, 2009. (cited on page 48)

[2] Carlos M Aderaldo, Nabor C Mendonça, Claus Pahl, and Pooyan Jamshidi. Benchmark requirements for microservices architecture research. In *International Workshop on Establishing the Community-Wide Infrastructure for Architecture-Based Software Engineering*, 2017. (cited on page 77)

[3] Ahmed M. Alaa, Scott Hu, and Mihaela van der Schaar. Learning from clinical judgments: Semi-Markov-modulated marked Hawkes processes for risk prognosis. In *International Conference on Machine Learning*, 2017. (cited on page 3)

[4] John Aldrich. RA Fisher and the making of maximum likelihood 1912-1922. *Statistical science*, 1997. (cited on page 7)

[5] Mahnoosh Alizadeh, Anna Scaglione, Jamie Davies, and Kenneth S Kurani. A scalable stochastic model for the electricity demand of electric and plug-in hybrid vehicles. *IEEE Transactions on Smart Grid*, 2013. (cited on page 69)

[6] Martín Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein generative adversarial networks. In *International Conference on Machine Learning*, 2017. (cited on page 56)

[7] François Baccelli and Jae Oh Woo. On the entropy and mutual information of point processes. In *International Symposium on Information Theory*, 2016. (cited on pages 11, 58)

[8] E. Bacry, M. Bompaire, S. Gaïffas, and S. Poulsen. tick: a Python library for statistical learning, with a particular emphasis on time-dependent modeling. *ArXiv preprint*, 2017. (cited on page 50)

[9] Emmanuel Bacry, Iacopo Mastromatteo, and Jean-François Muzy. Hawkes processes in finance. *Market Microstructure and Liquidity*, 2015. (cited on page 3)

[10] GA Barnard. Time intervals between accidents—a note on Maguire, Pearson and Wynn's paper. *Biometrika*, 1953. (cited on page 68)

[11] Federico Bergamin, Pierre-Alexandre Mattei, Jakob Drachmann Havtorn, Hugo Senetaire, Hugo Schmutz, Lars Maaløe, Soren Hauberg, and Jes Frellsen. Model-agnostic out-of-distribution detection using combined statistical tests. In *International Conference on Artificial Intelligence and Statistics*, 2022. (cited on page 83)

[12] Mark Berman. Inhomogeneous and modulated gamma processes. *Biometrika*, 1981. (cited on page 48)

[13] Marin Biloš, Johanna Sommer, Syama Sundar Rangapuram, Tim Januschowski, and Stephan Günnemann. Neural flows: Efficient alternative to neural ODEs. *Advances in Neural Information Processing Systems*, 2021. (cited on page 82)

[14] Eli Bingham, Jonathan P. Chen, Martin Jankowiak, Fritz Obermeyer, Neeraj Pradhan, Theofanis Karaletsos, Rohit Singh, Paul Szerlip, Paul Horsfall, and Noah D. Goodman. Pyro: Deep Universal Probabilistic Programming. *Journal of Machine Learning Research*, 2018. (cited on pages 118, 125)

[15] Christopher M Bishop. Mixture density networks. 1994. (cited on pages 27, 33, and 37)

[16] Christopher M Bishop and Nasser M Nasrabadi. *Pattern recognition and machine learning.* 2006. (cited on pages 7, 110)

[17] David M Blei, Alp Kucukelbir, and Jon D McAuliffe. Variational inference: A review for statisticians. *Journal of the American statistical Association*, 2017. (cited on page 56)

[18] Guy E Blelloch. Prefix sums and their applications. Technical report, 1990. (cited on page 46)

[19] Avrim Blum, John Hopcroft, and Ravindran Kannan. *Foundations of data science.* 2016. (cited on page 139)

[20] Vladimir Bogachev, Aleksandr Kolesnikov, and Kirill Medvedev. Triangular transformations of measures. *Sbornik: Mathematics*, 2005. (cited on page 44)

[21] Andrew Bray and Frederic Paik Schoenberg. Assessment of point process models for earthquake forecasting. *Statistical science*, 2013. (cited on page 7)

[22] Emery N Brown, Riccardo Barbieri, Valérie Ventura, Robert E Kass, and Loren M Frank. The time-rescaling theorem and its application to neural spike train data analysis. *Neural computation*, 2002. (cited on pages 15, 66, and 73)

[23] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter,

Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In *Advances in Neural Information Processing Systems*, 2020. (cited on pages 3, 7)

[24] Erhan Çinlar. *Introduction to Stochastic Processes.* 1975. (cited on page 48)

[25] Oscar Celma. Music recommendation. In *Music recommendation and discovery.* 2010. (cited on page 119)

[26] Bertrand Charpentier, Marin Biloš, and Stephan Günnemann. Uncertainty on asynchronous time event prediction. In *Advances in Neural Information Processing Systems*, 2019. (cited on pages 37, 49)

[27] Bertrand Charpentier, Daniel Zügner, and Stephan Günnemann. Posterior network: Uncertainty estimation without OOD samples via density-based pseudo-counts. In *Advances in Neural Information Processing Systems*, 2020. (cited on page 83)

[28] Ricky T. Q. Chen, Brandon Amos, and Maximilian Nickel. Learning neural event functions for ordinary differential equations. In *International Conference on Learning Representations*, 2021. (cited on pages 55, 82)

[29] Ricky T. Q. Chen, Brandon Amos, and Maximilian Nickel. Neural spatio-temporal point processes. In *International Conference on Learning Representations*, 2021. (cited on page 83)

[30] Tian Qi Chen, Yulia Rubanova, Jesse Bettencourt, and David Duvenaud. Neural ordinary differential equations. In *Advances in Neural Information Processing Systems*, 2018. (cited on page 81)

[31] Kyunghyun Cho, Bart van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder–decoder approaches. In *Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation*, 2014. (cited on page 24)

[32] Hyunsun Choi, Eric Jang, and Alexander A Alemi. WAIC, but why? Generative ensembles for robust anomaly detection. *ArXiv preprint*, 2018. (cited on pages 67, 72, and 139)

[33] Neil A Chriss and Neil Chriss. *Black–Scholes and beyond: Option pricing models.* 1997. (cited on page 61)

[34] Kacper Chwialkowski, Heiko Strathmann, and Arthur Gretton. A kernel test of goodness of fit. In *International Conference on Machine Learning*, 2016. (cited on page 73)

[35] David R Cox. Some statistical methods connected with series of events. *Journal of the Royal Statistical Society: Series B (Methodological)*, 1955. (cited on pages 3, 48, and 73)

[36] David Roxbee Cox. The statistical analysis of series of events. *Monographs on Applied Probability and Statistics*, 1966. (cited on pages 48, 69)

[37] David Roxbee Cox. The statistical analysis of dependencies in point processes. *Stochastic Point Processes. Wiley: New York*, 1972. (cited on pages 46, 48)

[38] David Roxbee Cox and Valerie Isham. *Point processes.* 1980. (cited on page 3)

[39] George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 1989. (cited on page 35)

[40] Ralph B D'Agostino. *Goodness-of-fit-techniques.* 1986. (cited on pages 66, 69, 75, and 140)

[41] Daryl J Daley and David Vere-Jones. *An introduction to the theory of point processes. Volume I: Elementary theory and methods.* 2003. (cited on pages 7, 69)

[42] Daryl J Daley and David Vere-Jones. *An introduction to the theory of point processes. Volume II: General theory and structure.* 2008. (cited on page 7)

[43] Hennie Daniels and Marina Velikova. Monotone and partially monotone neural networks. *IEEE Transactions on Neural Networks*, 2010. (cited on page 35)

[44] Anirban DasGupta. *Asymptotic theory of statistics and probability.* 2008. (cited on page 35)

[45] Angelos Dassios and Hongbiao Zhao. Exact simulation of Hawkes process with exponentially decaying intensity. *Electronic Communications in Probability*, 2013. (cited on pages 14, 45)

[46] Peter Dayan, Laurence F Abbott, et al. Theoretical neuroscience: Computational and mathematical modeling of neural systems. *Journal of Cognitive Neuroscience*, 2003. (cited on pages 3, 7, and 84)

[47] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Fei-Fei Li. ImageNet: A large-scale hierarchical image database. In *Conference on Computer Vision and Pattern Recognition*, 2009. (cited on page 84)

[48] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Annual Conference of the North American Chapter of the Association for Computational Linguistics*, 2019. (cited on pages 3, 84)

[49] Luc Devroye. Nonuniform random variate generation. *Handbooks in operations research and management science*, 1986. (cited on pages 12, 15, 18, and 20)

[50] Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. Density estimation using Real NVP. In *International Conference on Learning Representations*, 2017. (cited on pages 30, 48, 116, and 117)

[51] Christian Donner and Manfred Opper. Efficient Bayesian inference of sigmoidal Gaussian Cox processes. *The Journal of Machine Learning Research*, 2018. (cited on page 48)

[52] Nan Du, Hanjun Dai, Rakshit Trivedi, Utkarsh Upadhyay, Manuel Gomez-Rodriguez, and Le Song. Recurrent marked temporal point processes: Embedding event history to vector. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2016. (cited on pages 4, 27, 28, 32, 33, 35, 36, 37, 39, 43, 45, 49, 51, 71, 81, 83, 114, 117, 119, and 122)

[53] Conor Durkan, Artur Bekasov, Iain Murray, and George Papamakarios. Neural spline flows. In *Advances in Neural Information Processing Systems*, 2019. (cited on pages 43, 47, 49, and 127)

[54] Emil Eirola and Amaury Lendasse. Gaussian mixture models for time series modelling, forecasting, and interpolation. In *International Symposium on Intelligent Data Analysis*, 2013. (cited on page 37)

[55] Jeffrey L Elman. Finding structure in time. *Cognitive science*, 1990. (cited on pages 24, 33)

[56] Joseph Enguehard, Dan Busbridge, Adam Bozson, Claire Woodcock, and Nils Hammerla. Neural temporal point processes for modelling electronic health records. In *Machine Learning for Health*, 2020. (cited on pages 3, 78, and 84)

[57] William Feller. On the integro-differential equations of purely discontinuous markoff processes. *Transactions of the American Mathematical Society*, 1940. (cited on page 3)

[58] William Feller. On the theory of stochastic processes, with particular reference to applications. In *Proceedings of the Berkeley Symposium on Mathematical Statistics and Probability*, 1949. (cited on page 3)

[59] William Feller. On semi-Markov processes. *Proceedings of the National Academy of Sciences of the United States of America*, 1964. (cited on page 61)

[60] Erik M Ferragut, Jason Laska, and Robert A Bridges. A new, principled approach to anomaly detection. In *International Conference on Machine Learning and Applications*, 2012. (cited on page 72)

[61] Wolfgang Fischer and Kathleen Meier-Hellstern. The Markov-modulated Poisson process cookbook. *Performance Evaluation*, 1993. (cited on pages 56, 60)

[62] Ronald Aylmer Fisher. Design of experiments. *British Medical Journal*, 1936. (cited on page 67)

[63] Ronald Aylmer Fisher. Answer to question 14 on combining independent tests of significance. 1948. (cited on pages 69, 147)

[64] Sylvia Frühwirth-Schnatter. *Finite mixture and Markov switching models.* 2006. (cited on page 31)

[65] Felipe Gerhard and Wulfram Gerstner. Rescaling, thinning or complementing? On goodness-of-fit procedures for point process models and generalized linear models. In *Advances in Neural Information Processing Systems*, 2010. (cited on page 73)

[66] Felipe Gerhard, Robert Haslinger, and Gordon Pipa. Applying the multivariate time-rescaling theorem to neural population models. *Neural computation*, 2011. (cited on pages 69, 73, and 143)

[67] Mathieu Germain, Karol Gregor, Iain Murray, and Hugo Larochelle. MADE: Masked autoencoder for distribution estimation. In *International Conference on Machine Learning*, 2015. (cited on page 48)

[68] Wulfram Gerstner, Werner M Kistler, Richard Naud, and Liam Paninski. *Neuronal dynamics: From single neurons to networks and models of cognition.* 2014. (cited on pages 3, 7, 82, and 84)

[69] Tilmann Gneiting and Matthias Katzfuss. Probabilistic forecasting. *Annual Review of Statistics and Its Application*, 2014. (cited on page 83)

[70] Tilmann Gneiting, Larissa I Stanberry, Eric P Grimit, Leonhard Held, and Nicholas A Johnson. Assessing probabilistic forecasts of multivariate quantities, with an application to ensemble predictions of surface winds. 2008. (cited on page 83)

[71] Manuel Gomez-Rodriguez and Isabel Valera. Learning with temporal point processes. *Tutorial at International Conference on Machine Learning*, 2018. (cited on page 7)

[72] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning.* 2016. (cited on pages 3, 23)

[73] Yves Grandvalet and Yoshua Bengio. Entropy regularization. *Semi-supervised learning*, 2006. (cited on page 58)

[74] Alex Graves. Generating sequences with recurrent neural networks. *ArXiv preprint*, 2013. (cited on page 37)

[75] M Greenwood. The statistical study of infectious diseases. *Journal of the Royal Statistical Society: Series A*, 1946. (cited on pages 69, 73)

[76] Arthur Gretton, Karsten M Borgwardt, Malte J Rasch, Bernhard Schölkopf, and Alexander Smola. A kernel two-sample test. *Journal of Machine Learning Research*, 2012. (cited on pages 50, 129, and 130)

[77] Ruocheng Guo, Jundong Li, and Huan Liu. INITIATOR: Noise-contrastive estimation for marked temporal point process. In *International Joint Conference on Artificial Intelligence*, 2018. (cited on page 37)

[78] Vinayak Gupta, Srikanta Bedathur, Sourangshu Bhattacharya, and Abir De. Learning temporal point processes with intermittent observations. In *International Conference on Artificial Intelligence and Statistics*, 2021. (cited on page 82)

[79] Alan G Hawkes. Spectra of some self-exciting and mutually exciting point processes. *Biometrika*, 1971. (cited on pages 3, 13, 36, 45, 47, and 50)

[80] Alan G Hawkes. Hawkes processes and their applications to finance: A review. *Quantitative Finance*, 2018. (cited on page 3)

[81] Shilin He, Jieming Zhu, Pinjia He, and Michael R Lyu. Experience report: System log analysis for anomaly detection. In *International Symposium on Software Reliability Engineering*, 2016. (cited on pages 65, 78)

[82] Agnes Helmstetter and Didier Sornette. Importance of direct and indirect triggered seismicity in the ETAS model of seismicity. *Geophysical Research Letters*, 2003. (cited on page 3)

[83] ID Hill. Approximating the distribution of Greenwood's statistic with Johnson distributions. *Journal of the Royal Statistical Society: Series A*, 1979. (cited on page 73)

[84] Marcel Hirt and Petros Dellaportas. Scalable Bayesian learning for state space models using variational inference with SMC samplers. In *International Conference on Artificial Intelligence and Statistics*, 2019. (cited on page 62)

[85] Jonathan Ho, Xi Chen, Aravind Srinivas, Yan Duan, and Pieter Abbeel. Flow++: Improving flow-based generative models with variational dequantization and architecture design. In *International Conference on Machine Learning*, 2019. (cited on page 30)

[86] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 1997. (cited on page 24)

[87] Weihua Hu, Matthias Fey, Hongyu Ren, Maho Nakata, Yuxiao Dong, and Jure Leskovec. Ogb-lsc: A large-scale challenge for machine learning on graphs. *ArXiv preprint*, 2021. (cited on page 84)

[88] Chin-Wei Huang, David Krueger, Alexandre Lacoste, and Aaron C. Courville. Neural autoregressive flows. In *International Conference on Machine Learning*, 2018. (cited on pages 30, 35)

[89] Hengguan Huang, Hao Wang, and Brian Mak. Recurrent Poisson process unit for speech recognition. In *AAAI Conference on Artificial Intelligence*, 2019. (cited on page 36)

[90] Valerie Isham and Mark Westcott. A self-correcting point process. *Stochastic processes and their applications*, 1979. (cited on pages 14, 36, 47, and 50)

[91] Priyank Jaini, Kira A. Selby, and Yaoliang Yu. Sum-of-squares polynomial flow. In *International Conference on Machine Learning*, 2019. (cited on pages 30, 35, 43, 44, and 48)

[92] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. In *International Conference on Learning Representations*, 2017. (cited on pages 32, 59, 61, and 82)

[93] Dominik Janzing, Kailash Budhathoki, Lenon Minorics, and Patrick Blöbaum. Causal structure based root cause analysis of outliers. *ArXiv preprint*, 2019. (cited on page 72)

[94] Stephen P Jenkins. Survival analysis. *Unpublished manuscript, Institute for Social and Economic Research, University of Essex, Colchester, UK*, 2005. (cited on page 9)

[95] Junteng Jia and Austin R. Benson. Neural jump stochastic differential equations. In *Advances in Neural Information Processing Systems*, 2019. (cited on pages 4, 81)

[96] Hiroshi Kajino. A differentiable point process with its application to spiking neural networks. In *International Conference on Machine Learning*, 2021. (cited on page 82)

[97] Song-Hee Kim and Ward Whitt. Are call center and hospital arrivals well modeled by nonhomogeneous Poisson processes? *Manufacturing & Service Operations Management*, 2014. (cited on page 69)

[98] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, 2015. (cited on pages 50, 120)

[99] Diederik P. Kingma and Max Welling. Auto-encoding variational Bayes. In *International Conference on Learning Representations*, 2014. (cited on pages 57, 61)

[100] Durk P Kingma, Tim Salimans, Rafal Jozefowicz, Xi Chen, Ilya Sutskever, and Max Welling. Improved variational inference with inverse autoregressive flow. In *Advances in Neural Information Processing Systems*, 2016. (cited on page 48)

[101] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. ImageNet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, 2012. (cited on page 3)

[102] Srijan Kumar, Xikun Zhang, and Jure Leskovec. Predicting dynamic embedding trajectory in temporal interaction networks. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2019. (cited on pages 37, 119)

[103] Thomas A. Lasko. Efficient inference of Gaussian-process-modulated renewal processes with application to medical event data. In *Conference on Uncertainty in Artificial Intelligence*, 2014. (cited on page 48)

[104] Alexei Ledenev et al. Pumba: Chaos testing tool for Docker. `https://github.com/alexei-led/pumba`, 2016. (cited on pages 77, 146)

[105] PA W Lewis and Gerald S Shedler. Simulation of nonhomogeneous Poisson processes by thinning. *Naval research logistics quarterly*, 1979. (cited on pages 3, 20, and 48)

[106] P.A.W. Lewis. *Stochastic Point Processes: Statistical Analysis, Theory and Applications*. 1972. (cited on page 3)

[107] Peter A. W. Lewis. Some results on tests for Poisson processes. *Biometrika*, 1965. (cited on page 73)

[108] Shuang Li, Shuai Xiao, Shixiang Zhu, Nan Du, Yao Xie, and Le Song. Learning temporal point processes via reinforcement learning. In *Advances in Neural Information Processing Systems*, 2018. (cited on pages 4, 17, 36, 37, 55, 57, 62, and 69)

[109] Shiyu Liang, Yixuan Li, and R. Srikant. Enhancing the reliability of out-of-distribution image detection in neural networks. In *International Conference on Learning Representations*, 2018. (cited on page 66)

[110] Haitao Lin, Cheng Tan, Lirong Wu, Zhangyang Gao, Stan Li, et al. An empirical study: Extensive deep temporal point process. *ArXiv preprint*, 2021. (cited on page 83)

[111] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft COCO: Common objects in context. In *European conference on computer vision*, 2014. (cited on page 84)

[112] Scott W. Linderman and Ryan P. Adams. Discovering latent network structure in point process data. In *International Conference on Machine Learning*, 2014. (cited on page 7)

[113] Qiang Liu, Jason D. Lee, and Michael I. Jordan. A kernelized Stein discrepancy for goodness-of-fit tests. In *International Conference on Machine Learning*, 2016. (cited on page 73)

[114] Siqi Liu and Milos Hauskrecht. Event outlier detection in continuous time. In *International Conference on Machine Learning*, 2021. (cited on pages 72, 83)

[115] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 2008. (cited on page 40)

[116] Chris J. Maddison, Andriy Mnih, and Yee Whye Teh. The concrete distribution: A continuous relaxation of discrete random variables. In *International Conference on Learning Representations*, 2017. (cited on pages 59, 61, and 82)

[117] George Marsaglia, Wai Wan Tsang, Jingbo Wang, and Others. Evaluating Kolmogorov's distribution. *Journal of Statistical Software*, 2003. (cited on page 72)

[118] Geoffrey McLachlan and David Peel. *Finite mixture models*. 2004. (cited on page 31)

[119] Hongyuan Mei and Jason Eisner. The neural Hawkes process: A neurally self-modulating multivariate point process. In *Advances in Neural Information Processing Systems*, 2017. (cited on pages 4, 27, 28, 35, 36, 37, 49, and 83)

[120] Hongyuan Mei, Chenghao Yang, and Jason Eisner. Transformer embeddings of irregularly spaced events and their participants. In *International Conference on Learning Representations*, 2021. (cited on page 81)

[121] Shakir Mohamed, Mihaela Rosca, Michael Figurnov, and Andriy Mnih. Monte Carlo gradient estimation in machine learning. *ArXiv preprint*, 2019. (cited on pages 32, 57, 60, 61, and 124)

[122] Kristen Moore, Cody J Christopher, David Liebowitz, Surya Nepal, and Renee Selvey. Modelling direct messaging networks with multiple recipients for cyber deception. *ArXiv preprint*, 2021. (cited on page 82)

[123] PAP Moran. The random division of an interval. *Supplement to the Journal of the Royal Statistical Society*, 1947. (cited on pages 140, 141)

[124] Lucy Morgan, Barry Nelson, Andrew Titman, and David Worthington. A spline-based method for modelling and generating a nonhomogeneous Poisson process. In *Winter Simulation Conference*, 2019. (cited on page 48)

[125] Warren R. Morningstar, Cusuh Ham, Andrew G. Gallagher, Balaji Lakshminarayanan, Alex Alemi, and Joshua V. Dillon. Density of states estimation for out of distribution detection. In *International Conference on Artificial Intelligence and Statistics*, 2021. (cited on page 72)

[126] S Mostafa Mousavi, Yixiao Sheng, Weiqiang Zhu, and Gregory C Beroza. STanford EArthquake Dataset (STEAD): A global data set of seismic signals for ai. *IEEE Access*, 2019. (cited on pages 3, 77, and 146)

[127] Kevin P Murphy. *Machine learning: A probabilistic perspective.* 2012. (cited on page 7)

[128] Eric Nalisnick, Akihiro Matsukawa, Yee Whye Teh, and Balaji Lakshminarayanan. Detecting out-of-distribution inputs to deep generative models using a test for typicality. *ArXiv preprint*, 2019. (cited on pages 65, 67, 70, 72, 75, 83, 139, and 140)

[129] Eric T. Nalisnick, Akihiro Matsukawa, Yee Whye Teh, Dilan Görür, and Balaji Lakshminarayanan. Do deep generative models know what they don't know? In *International Conference on Learning Representations*, 2019. (cited on page 83)

[130] Jerzy Neyman and Egon Sharpe Pearson. On the problem of the most efficient tests of statistical hypotheses. *Philosophical Transactions of the Royal Society of London: Series A*, 1933. (cited on page 78)

[131] Bernard V North, David Curtis, and Pak C Sham. A note on the calculation of empirical p-values from Monte Carlo procedures. *The American Journal of Human Genetics*, 2002. (cited on pages 71, 143)

[132] David Oakes. The Markovian self-exciting process. *Journal of Applied Probability*, 1975. (cited on pages 13, 45)

[133] Yosihiko Ogata. On Lewis' simulation method for point processes. *Transactions on Information Theory*, 1981. (cited on pages 3, 14, and 20)

[134] Yosihiko Ogata. Statistical models for earthquake occurrences and residual analysis for point processes. *Journal of the American Statistical Association*, 1988. (cited on pages 3, 14, 73, and 84)

[135] Yosihiko Ogata. Seismicity analysis through point-process modeling: A review. *Seismicity patterns, their statistical significance and physical meaning*, 1999. (cited on pages 3, 84)

[136] Yosihiko Ogata. Statistics of earthquake activity: Models and methods for earthquake predictability studies. *Annual Review of Earth and Planetary Sciences*, 2017. (cited on page 84)

[137] César Ali Marin Ojeda, Kostadin Cvejoski, Rafet Sifa, Jannis Schuecker, and Christian Bauckhage. Patterns and outliers in temporal point processes. In *SAI Intelligent Systems Conference*, 2019. (cited on page 72)

[138] Maya Okawa, Tomoharu Iwata, Takeshi Kurashima, Yusuke Tanaka, Hiroyuki Toda, and Naonori Ueda. Deep mixture point processes: Spatio-temporal event

prediction with rich contextual information. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2019. (cited on page 36)

[139] Takahiro Omi, Naonori Ueda, and Kazuyuki Aihara. Fully neural network based model for general temporal point processes. In *Advances in Neural Information Processing Systems*, 2019. (cited on pages 27, 28, 35, 36, 37, 43, 45, 49, 50, 81, 83, 114, 117, 118, and 127)

[140] Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. *ArXiv preprint*, 2016. (cited on pages 37, 48)

[141] Manfred Opper and Guido Sanguinetti. Variational inference for Markov jump processes. In *Advances in Neural Information Processing Systems*, 2007. (cited on pages 55, 62)

[142] George Papamakarios, Iain Murray, and Theo Pavlakou. Masked autoregressive flow for density estimation. In *Advances in Neural Information Processing Systems*, 2017. (cited on page 48)

[143] George Papamakarios, Eric Nalisnick, Danilo Jimenez Rezende, Shakir Mohamed, and Balaji Lakshminarayanan. Normalizing flows for probabilistic modeling and inference. *ArXiv preprint*, 2019. (cited on pages 47, 48)

[144] Raghu Pasupathy. Generating homogeneous Poisson processes. *Wiley encyclopedia of operations research and management science*, 2010. (cited on pages 48, 68)

[145] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. PyTorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*, 2019. (cited on pages 49, 116, and 125)

[146] Jonathan W. Pillow. Time-rescaling methods for the estimation and assessment of non-Poisson neural encoding models. In *Advances in Neural Information Processing Systems*, 2009. (cited on page 69)

[147] Rajesh Ranganath, Sean Gerrish, and David M. Blei. Black box variational inference. In *International Conference on Artificial Intelligence and Statistics*, 2014. (cited on page 57)

[148] JS Rao and Morgan Kuo. Asymptotic results on the Greenwood statistic and some of its generalizations. *Journal of the Royal Statistical Society: Series B*, 1984. (cited on page 73)

[149] Vinayak Rao and Yee Whye Teh. Fast MCMC sampling for Markov jump processes and extensions. *The Journal of Machine Learning Research*, 2013. (cited on pages 62, 63, and 138)

[150] Vinayak A. Rao and Yee Whye Teh. Gaussian process modulated renewal processes. In *Advances in Neural Information Processing Systems*, 2011. (cited on page 48)

[151] Vinayak A. Rao and Yee Whye Teh. MCMC for continuous-time discrete-state systems. In *Advances in Neural Information Processing Systems*, 2012. (cited on page 133)

[152] Jakob Gulddahl Rasmussen. Temporal point processes. *Lecture Notes*, 2011. (cited on pages 7, 33, 36, 52, and 113)

[153] Jakob Gulddahl Rasmussen. Lecture notes: Temporal point processes and the conditional intensity function. *ArXiv preprint*, 2018. (cited on page 71)

[154] Jie Ren, Peter J. Liu, Emily Fertig, Jasper Snoek, Ryan Poplin, Mark A. DePristo, Joshua V. Dillon, and Balaji Lakshminarayanan. Likelihood ratios for out-of-distribution detection. In *Advances in Neural Information Processing Systems*, 2019. (cited on pages 65, 67, and 72)

[155] Danilo Jimenez Rezende and Shakir Mohamed. Variational inference with normalizing flows. In *International Conference on Machine Learning*, 2015. (cited on pages 27, 30, and 37)

[156] Danilo Jimenez Rezende, Daan Wierstra, and Wulfram Gerstner. Variational learning for recurrent spiking networks. In *Advances in Neural Information Processing Systems*, 2011. (cited on page 82)

[157] Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. Stochastic backpropagation and approximate inference in deep generative models. In *International Conference on Machine Learning*, 2014. (cited on pages 57, 61)

[158] Marian-Andrei Rizoiu, Young Lee, Swapnil Mishra, and Lexing Xie. A tutorial on Hawkes processes for events in social media. *ArXiv preprint*, 2017. (cited on page 14)

[159] Yulia Rubanova, Tian Qi Chen, and David Duvenaud. Latent ordinary differential equations for irregularly-sampled time series. In *Advances in Neural Information Processing Systems*, 2019. (cited on page 81)

[160] Reuven Y Rubinstein. *Monte Carlo optimization, simulation, and sensitivity of queueing networks.* 1986. (cited on page 61)

[161] Lukas Ruff, Jacob R Kauffmann, Robert A Vandermeulen, Grégoire Montavon, Wojciech Samek, Marius Kloft, Thomas G Dietterich, and Klaus-Robert Müller.

A unifying review of deep and shallow anomaly detection. *Proceedings of the IEEE*, 2021. (cited on pages 67, 72)

[162] William H Savran, Maximilian J Werner, Warner Marzocchi, David A Rhoades, David D Jackson, Kevin Milner, Edward Field, and Andrew Michael. Pseudo-prospective evaluation of UCERF3-ETAS forecasts during the 2019 ridgecrest sequence. *Bulletin of the Seismological Society of America*, 2020. (cited on page 83)

[163] Anton Maximilian Schäfer and Hans Georg Zimmermann. Recurrent neural networks are universal approximators. In *International Conference on Artificial Neural Networks*, 2006. (cited on page 35)

[164] Mike Schuster. Better generative models for sequential data problems: Bidirectional recurrent mixture density networks. In *Advances in Neural Information Processing Systems*, 2000. (cited on page 37)

[165] SeedScientific. How much data is created every day? `https://seedscientific.com/how-much-data-is-created-every-day/`. Accessed: 2022-04-06. (cited on page 3)

[166] Karishma Sharma, Yizhou Zhang, Emilio Ferrara, and Yan Liu. Identifying coordinated accounts on social media through hidden influence and group behaviours. 2020. (cited on pages 81, 82)

[167] Oleksandr Shchur, Marin Biloš, and Stephan Günnemann. Intensity-free learning of temporal point processes. In *International Conference on Learning Representations*, 2020. (cited on pages 5, 43, 45, 49, 50, 75, 83, and 146)

[168] Oleksandr Shchur, Nicholas Gao, Marin Biloš, and Stephan Günnemann. Fast and flexible temporal point processes with triangular maps. In *Advances in Neural Information Processing Systems*, 2020. (cited on pages 5, 55)

[169] Oleksandr Shchur, Ali Caner Türkmen, Tim Januschowski, Jan Gasthaus, and Stephan Günnemann. Detecting anomalous event sequences with temporal point processes. *Advances in Neural Information Processing Systems*, 2021. (cited on pages 5, 7)

[170] Oleksandr Shchur, Ali Caner Türkmen, Tim Januschowski, and Stephan Günnemann. Neural temporal point processes: A review. *International Joint Conference on Artificial Intelligence*, 2021. (cited on pages 3, 5, 7, 65, and 83)

[171] Hava T Siegelmann and Eduardo D Sontag. On the computational power of neural nets. In *Proceedings of the fifth annual workshop on Computational learning theory*, 1992. (cited on page 35)

[172] Walter L Smith. Renewal theory and its ramifications. *Journal of the Royal Statistical Society: Series B (Methodological)*, 1958. (cited on page 12)

[173] Markus Steinbach, Anshul Jindal, Mohak Chadha, Michael Gerndt, and Shajulin Benedict. Tppfaas: Modeling serverless functions invocations via temporal point processes. *IEEE Access*, 2022. (cited on page 82)

[174] Michael A Stephens. Further percentage points for Greenwood's statistic. *Journal of the Royal Statistical Society: Series A*, 1981. (cited on page 73)

[175] Olav Stetter, Demian Battaglia, Jordi Soriano, and Theo Geisel. Model-free reconstruction of excitatory neuronal connectivity from calcium imaging signals. *PLoS Computational Biology*, 2012. (cited on page 76)

[176] Esteban G Tabak and Cristina V Turner. A family of nonparametric density estimation algorithms. *Communications on Pure and Applied Mathematics*, 2013. (cited on pages 30, 37)

[177] Behzad Tabibian, Isabel Valera, Mehrdad Farajtabar, Le Song, Bernhard Schölkopf, and Manuel Gomez-Rodriguez. Distilling information reliability and source trustworthiness from digital traces. In *Proceedings of the Web Conference*, 2017. (cited on page 36)

[178] Matthew A Taddy, Athanasios Kottas, et al. Mixture modeling for marked Poisson processes. *Bayesian Analysis*, 2012. (cited on page 36)

[179] Long Tao, Karoline E Weber, Kensuke Arai, and Uri T Eden. A common goodness-of-fit framework for neural population models using marked point process time-rescaling. *Journal of Computational Neuroscience*, 2018. (cited on pages 69, 73, and 143)

[180] Lucas Theis, Aäron van den Oord, and Matthias Bethge. A note on the evaluation of generative models. In *International Conference on Learning Representations*, 2016. (cited on pages 50, 83)

[181] Ali Caner Türkmen, Yuyang Wang, and Alexander J Smola. FastPoint: Scalable deep point processes. In *ECML-PKDD*, 2019. (cited on pages 4, 37, 48, 49, and 50)

[182] Utkarsh Upadhyay and Manuel Gomez Rodriguez. Temporal point processes. Lecture notes for Human-Centered ML, 2019. (cited on page 35)

[183] Utkarsh Upadhyay, Abir De, and Manuel Gomez Rodriguez. Deep reinforcement learning of marked temporal point processes. In *Advances in Neural Information Processing Systems*, 2018. (cited on pages 32, 36, 37, 55, 56, 57, 62, 113, 117, and 121)

[184] Aäron van den Oord, Nal Kalchbrenner, Lasse Espeholt, Koray Kavukcuoglu, Oriol Vinyals, and Alex Graves. Conditional image generation with PixelCNN decoders. In *Advances in Neural Information Processing Systems*, 2016. (cited on page 48)

[185] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, 2017. (cited on pages 24, 43, 49, and 81)

[186] Athanasios Voulodimos, Nikolaos Doulamis, Anastasios Doulamis, and Eftychios Protopapadakis. Deep learning for computer vision: A brief review. *Computational intelligence and neuroscience*, 2018. (cited on page 3)

[187] Martin J Wainwright, Michael I Jordan, et al. Graphical models, exponential families, and variational inference. *Foundations and Trends in Machine Learning*, 2008. (cited on pages 56, 58)

[188] Pengyu Wang and Phil Blunsom. Collapsed variational Bayesian inference for hidden Markov models. In *International Conference on Artificial Intelligence and Statistics*, 2013. (cited on pages 61, 136)

[189] Ziyu Wang, Bin Dai, David P. Wipf, and Jun Zhu. Further analysis of outlier detection with deep generative models. In *Advances in Neural Information Processing Systems*, 2020. (cited on pages 65, 70, and 72)

[190] Weave. Sock shop : A microservice demo application. `https://github.com/microservices-demo/microservices-demo`, 2017. (cited on pages 77, 146)

[191] Song Wei, Shixiang Zhu, Minghe Zhang, and Yao Xie. Goodness-of-fit test for mismatched self-exciting processes. In *International Conference on Artificial Intelligence and Statistics*, 2021. (cited on page 73)

[192] Malcolm CA White, Yehuda Ben-Zion, and Frank L Vernon. A detailed earthquake catalog for the San Jacinto fault-zone region in southern California. *Journal of Geophysical Research: Solid Earth*, 2019. (cited on page 3)

[193] Andreas Wienke. *Frailty models in survival analysis.* 2010. (cited on page 113)

[194] Christian Wildner and Heinz Koeppl. Moment-based variational inference for Markov jump processes. In *International Conference on Machine Learning*, 2019. (cited on page 62)

[195] Alex H. Williams, Anthony Degleris, Yixin Wang, and Scott W. Linderman. Point process models for sequence detection in high-dimensional neural spike trains. In *Advances in Neural Information Processing Systems*, 2020. (cited on page 84)

[196] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 1992. (cited on pages 57, 61)

[197] Jing Wu, Owen Ward, James Curley, and Tian Zheng. Markov-modulated Hawkes processes for sporadic and bursty event occurrences. *ArXiv preprint*, 2019. (cited on page 62)

[198] Qitian Wu, Chaoqi Yang, Hengrui Zhang, Xiaofeng Gao, Paul Weng, and Guihai Chen. Adversarial training model unifying feature driven and point process perspectives for event popularity prediction. In *International Conference on Information and Knowledge Management*, 2018. (cited on pages 4, 17, 55, 56, and 62)

[199] Shuai Xiao, Mehrdad Farajtabar, Xiaojing Ye, Junchi Yan, Xiaokang Yang, Le Song, and Hongyuan Zha. Wasserstein learning of deep generative point process models. In *Advances in Neural Information Processing Systems*, 2017. (cited on pages 4, 37, 51, 55, 56, 62, and 130)

[200] Shuai Xiao, Junchi Yan, Xiaokang Yang, Hongyuan Zha, and Stephen M. Chu. Modeling the intensity function of point process via recurrent neural networks. In *AAAI Conference on Artificial Intelligence*, 2017. (cited on page 4)

[201] Shuai Xiao, Hongteng Xu, Junchi Yan, Mehrdad Farajtabar, Xiaokang Yang, Le Song, and Hongyuan Zha. Learning conditional generative models for temporal point processes. In *AAAI Conference on Artificial Intelligence*, 2018. (cited on page 37)

[202] Junchi Yan, Xin Liu, Liangliang Shi, Changsheng Li, and Hongyuan Zha. Improving maximum likelihood estimation of temporal point process via discriminative and adversarial learning. In *International Joint Conference on Artificial Intelligence*, 2018. (cited on pages 17, 37, 55, 56, and 62)

[203] Jiasen Yang, Vinayak A. Rao, and Jennifer Neville. A Stein-Papangelou goodness-of-fit test for point processes. In *International Conference on Artificial Intelligence and Statistics*, 2019. (cited on page 73)

[204] Jiahui Yu, Zhe Lin, Jimei Yang, Xiaohui Shen, Xin Lu, and Thomas S. Huang. Generative image inpainting with contextual attention. In *Conference on Computer Vision and Pattern Recognition*, 2018. (cited on page 7)

[205] Boqian Zhang and Vinayak Rao. Efficient parameter sampling for Markov jump processes. *ArXiv preprint*, 2017. (cited on page 62)

[206] Boqian Zhang, Jiangwei Pan, and Vinayak A. Rao. Collapsed variational Bayes for Markov jump processes. In *Advances in Neural Information Processing Systems*, 2017. (cited on pages 61, 62)

[207] Cheng Zhang, Judith Bütepage, Hedvig Kjellström, and Stephan Mandt. Advances in variational inference. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2018. (cited on pages 48, 61, and 135)

[208] Qiang Zhang, Aldo Lipani, Ömer Kirnap, and Emine Yilmaz. Self-attentive Hawkes process. In *International Conference on Machine Learning*, 2020. (cited on pages 43, 49, 81, and 83)

[209] Qiang Zhang, Aldo Lipani, and Emine Yilmaz. Learning neural point processes with latent graphs. In *Proceedings of the Web Conference*, 2021. (cited on page 83)

[210] Wei Zhang, Thomas Kobber Panum, Somesh Jha, Prasad Chalasani, and David Page. CAUSE: Learning Granger causality from event sequences using attribution methods. In *International Conference on Machine Learning*, 2020. (cited on page 83)

[211] Yizhou Zhang, Karishma Sharma, and Yan Liu. Vigdet: Knowledge informed neural temporal point process for coordination detection on social media. *Advances in Neural Information Processing Systems*, 2021. (cited on page 82)

[212] Ziwei Zhang, Peng Cui, and Wenwu Zhu. Deep learning on graphs: A survey. *Transactions on Knowledge and Data Engineering*, 2020. (cited on page 3)

[213] Zihao Zhou, Xingyi Yang, Ryan Rossi, Handong Zhao, and Rose Yu. Neural point process for learning spatiotemporal event dynamics. In *Learning for Dynamics and Control Conference*, 2022. (cited on page 83)

[214] Shixiang Zhu, Henry Shaowu Yuchi, and Yao Xie. Adversarial anomaly detection for marked spatio-temporal streaming data. In *International Conference on Acoustics, Speech and Signal Processing*, 2020. (cited on pages 72, 78)

[215] Shixiang Zhu, Minghe Zhang, Ruyi Ding, and Yao Xie. Deep fourier kernel for self-attentive point processes. In *International Conference on Artificial Intelligence and Statistics*, 2021. (cited on pages 49, 81, and 83)

[216] Zachary M. Ziegler and Alexander M. Rush. Latent normalizing flows for discrete sequences. In *International Conference on Machine Learning*, 2019. (cited on page 37)

[217] Simiao Zuo, Haoming Jiang, Zichong Li, Tuo Zhao, and Hongyuan Zha. Transformer Hawkes process. In *International Conference on Machine Learning*, 2020. (cited on pages 43, 49, 81, and 83)

A Notation

General

- $\mathbb{R}^D$ — D-dimensional Euclidean space.
- $\mathbb{R}_+ = (0, \infty)$ — set of strictly positive real numbers.
- $[0, T] \subset \mathbb{R}$ — time interval on which the TPP is observed.
- $t_i \in [0, T]$ — arrival time of the ith event.
- $\mathcal{M}$ — set of possible values that a mark can take (usually $\{1, \dots, C\}$ or $\mathbb{R}^D$).
- $m_i \in \mathcal{M}$ — mark corresponding to the ith event.
- $\tau_i = t_i - t_{i-1}$ — the ith inter-event time (assuming $t_0 = 0$ and $t_{N+1} = T$).
- $\mathcal{T} = (t_1, ..., t_N)$ — realization of a TPP represented by a sequence of strictly increasing arrival times.
- N — number of events in a TPP realization.
- $\mathcal{H}(t) = \{t_j \in \mathcal{T} : t_j < t\}$ — history consisting of events observed before time t.
- dt — infinitesimal change in t. Corresponds to a shortcut $g(dt) = \lim_{\Delta t \to 0} \frac{g(\Delta t)}{\Delta t}$ for any function g.
- $P_i^*(t|t_1, \dots, t_{i-1})$ — probability distribution over the next arrival time t_i given past events. This is always a continuous distribution supported on $[t_{i-1}, \infty)$.
- $p_i^*(t) = p_i(t|t_1, ..., t_{i-1})$ — probability density function (PDF) of the ith event (Equation 2.3).
- $F_i^*(t) = F_i(t|t_1, ..., t_{i-1})$ — cumulative distribution function (CDF) of the ith event (Equation 2.5).
- $S_i^*(t) = S_i(t|t_1, ..., t_{i-1})$ — survival function (SF) of the ith event (Equation 2.6).
- $\phi_i^*(t) = \phi_i(t|t_1, ..., t_{i-1})$ — hazard function (HF) of the ith event (Equation 2.7).
- $\Phi_i^*(t) = \Phi_i(t|t_1, ..., t_{i-1})$ — cumulative hazard function (CHF) of the ith event (Equation 2.8).
- $\lambda^*(t) = \lambda(t|\mathcal{H}(t))$ — conditional intensity at time t (Equation 2.9).

- $\Lambda^*(t) = \Lambda(t|t_1, ..., t_{i-1})$ — cumulative conditional intensity at time t, also known as the compensator (Equation 2.20).
- $\mathcal{Z} = (z_1, ..., z_N)$ — arrival times transformed by the compensator as $z_i = \Lambda^*(t_i)$ (see Theorem 1).
- $p(\mathcal{T})$ — likelihood, also known as point process density (Equation 2.22). We use $p(\mathcal{T})$ to refer both to the density function as well as the corresponding TPP .
- $\boldsymbol{\theta}$ — parameters of a TPP model. We implicitly assume that all above functions $\lambda^*(t)$, $\Lambda^*(t)$, $p_i^*(t)$, $F_i^*(t)$, $S_i^*(t)$, $\phi_i^*(t)$, $\Phi_i^*(t)$, $p(\mathcal{T})$ are parametrized by $\boldsymbol{\theta}$.
- $\mathbb{1}(x)$ — indicator function defined as

$$\mathbb{1}(x) = \begin{cases} 1 & \text{if } x \text{ is True,} \\ 0 & \text{else.} \end{cases}$$

- C — number of event types in a marked TPP with categorical marks.
- $\lambda_c^*(t)$ — conditional intensity of mark $c \in \{1, \dots, C\}$ in a marked TPP with categorical marks (Equation 2.30).

Chapter 3

- g — bijective differentiable transformation that specifies a normalizing flow model (Equation 3.5).
- $f_{\boldsymbol{\psi}}$ — individual parametric functions that are composed to define g.
- $\sigma(x) = 1/(1 + e^{-x})$ — the sigmoid function.
- $\boldsymbol{h}_i$ — history embedding computed after observing events $(t_1, \dots, t_{i-1})$.
- $\boldsymbol{y}_i$ — additional conditional information for each event.
- $\boldsymbol{e}_j$ — learnable embedding vector for the event sequence.
- $\boldsymbol{c}_i$ — context vector obtained by concatenating $\boldsymbol{h}_i$, $\boldsymbol{y}_i$, and $\boldsymbol{e}_j$.
- $\boldsymbol{\varphi}_i$ — parameters of the conditional distribution $p_i^*(\tau|\boldsymbol{\varphi}_i)$.
- $\boldsymbol{w} \in \Delta^{K-1}$, $\boldsymbol{\mu} \in \mathbb{R}^K$, $\boldsymbol{s} \in \mathbb{R}_+^K$ — parameters of the log-normal mixture distribution (Equation 3.8).
- K — number of components for DSF, SOS and LogNormMix models. Higher value corresponds to a more flexible distribution (potentially with more modes).
- H — dimension of the history embedding $\boldsymbol{h}_i$.

Chapter 4

- $\tilde{p}(\mathcal{Z})$ — density of the standard Poisson process (SPP).
- $\boldsymbol{F} = (f_1, ..., f_N)$ — increasing lower-triangular map that converts a realization $\mathcal{T}$ of a TPP with compensator Λ^* into a sample $\mathcal{Z}$ from the SPP.
- $f_i(t_1, ..., t_i) = \Lambda^*(t_i) = \Lambda(t_i | t_1, ..., t_{i-1})$ — component function of the map $\boldsymbol{F}$.
- $\boldsymbol{F}^{-1}$ — inverse of the map $\boldsymbol{F}$ defined above. $\boldsymbol{F}^{-1}$ converts a sample $\mathcal{Z} = (z_1, \dots, z_N)$ from the SPP into a realization $\mathcal{T} = (\Lambda^{*-1}(z_1), \dots, \Lambda^{*-1}(t_N))$ from a TPP with compensator Λ^*.
- $\Lambda \colon [0, T] \to \mathbb{R}_+$ — Cumulative intensity of an inhomogeneous Poisson process. (Note the difference from the compensator Λ^* of a general TPP that depends on the history, as indicated by $*$.)
- $\boldsymbol{\Lambda}$ — a map that applies the function Λ elementwise to a variable-length sequence $(t_1, \dots, t_N)$.
- $\Phi \colon \mathbb{R}_+ \to \mathbb{R}_+$ — cumulative hazard function of a renewal process. (Note the difference from the conditional CHF Φ_i^* of a general TPP that is different for each i and is history-dependent.)
- $\boldsymbol{\Phi}$ — a map that applies the function Φ elementwise to a variable-length sequence $(\tau_1, \dots, \tau_{N+1})$.
- $\boldsymbol{C}$ — the $N \times N$ cumulative sum matrix,
$$C_{ij} = \begin{cases} 1 & \text{if } i \leq j, \\ 0 & \text{else.} \end{cases}$$
- $\boldsymbol{D} \equiv \boldsymbol{C}^{-1}$ — the $N \times N$ difference matrix,
$$D_{ij} = \begin{cases} 1 & \text{if } i = j, \\ -1 & \text{if } i = j + 1, \\ 0 & \text{else.} \end{cases}$$
- $\boldsymbol{B}$ — an $N \times N$ block-diagonal matrix, where each block is a repeated $H \times H$ lower-triangular matrix with strictly positive diagonal entries.

Chapter 5

- $\boldsymbol{F}_{\boldsymbol{\theta}}^{-1}$ — a map that converts a sample $\mathcal{Z}$ from the SPP into a realization $\mathcal{T}$ of a TPP with compensator Λ^*. Defined same as in Chapter 4, but now we make the dependence on parameters $\boldsymbol{\theta}$ explicit.

- S — number of Monte Carlo samples used to approximate the expectation.
- $\zeta > 0$ — temperature parameter for the differentiable relaxation.
- $\sigma_\zeta(x) = 1/(1+\exp(-x/\zeta))$ — sigmoid function with temperature parameter ζ. As $\zeta \to 0$, the sigmoid function $\sigma_\zeta(x)$ approaches the indicator function $\mathbb{1}(x > 0)$.
- K — number of states in a Markov jump process.
- $(\mathcal{T}, \mathcal{S})$ — a Markov jump process trajectoriy represented by the jump times $\mathcal{T}$ and the sequence of visited states $\mathcal{S}$.
- $\mathcal{X}$ — observed events in a Markov modulated Poisson process.
- $q(\mathcal{T}, \mathcal{S})$ — approximate posterior distribution over jump times and visited states.
- $\boldsymbol{\psi} = \{\boldsymbol{\pi}, \boldsymbol{A}, \boldsymbol{\lambda}\}$ — parameters of the Markov modulated Poisson process (see Appendix F.1.1).

Chapter 6

- $X = (t_1, ..., t_N)$ — realization of a TPP (was denoted by $\mathcal{T}$ in preceding chapters).
- $\mathbb{P}$ — a TPP (i.e., a probability distribution over event sequences).
- $\mathcal{X}$ — set of variable-length event sequences (i.e., sample space of a TPP).
- H_0 and H_1 — null hypothesis and alternate hypothesis.
- $\mathbb{P}_{\text{data}}$ — probability distribution that generated the training data.
- $\mathbb{P}_{\text{model}}$ — probability distribution corresponding to the fitted model.
- $\mathbb{Q}$ — probability distribution corresponding to the alternate hypothesis.
- $\mathcal{D}_{\text{train}} = (X_1, \dots, X_M)$ — training set consisting of samples generated i.i.d. from some distribution $\mathbb{P}_{\text{data}}$.
- $\mathcal{D}_{\text{test}} = \mathcal{D}_{\text{test}}^{\text{ID}} \cup \mathcal{D}_{\text{test}}^{\text{OOD}}$ — test set consisting of in-distribution sequences $\mathcal{D}_{\text{test}}^{\text{ID}}$ and out-of-distribution sequences $\mathcal{D}_{\text{test}}^{\text{OOD}}$.
- δ — detectability parameter, higher values make the distributions $\mathbb{P}$ and $\mathbb{Q}$ more dissimilar.
- $s \colon \mathcal{X} \to \mathbb{R}$ — test statistic.
- p_s — p-value computed based on the test statistic s.
- α — confidence level (i.e., desired false positive rate).

- $Z = (v_1, ..., v_N) = (\Lambda^*(t_1), ..., \Lambda^*(t_N))$ — event sequence X converted using the compensator (was denoted by $\mathcal{Z} = (z_1, ..., z_N)$ in preceeding chapters).
- $V = \Lambda^*(T)$ — length of the transformed interval in random time change theorem.
- $w_i = v_i - v_{i-1}$ — inter-event times in the transformed sequence Z (assuming $v_0 = 0$ and $v_{N+1} = V$).
- κ_{arr} — Kolmogorov–Smirnov statistic for the arrival times (Equation 6.5).
- κ_{int} — Kolmogorov–Smirnov statistic for the inter-event times (Equation 6.6).
- ψ — sum-of-squared-spacings (3S) statistic (Equation 6.7).

B Abbreviations

General

- CDF — cumulative distribution function (Equation 2.5).
- CHF — cumulative hazard function (Equation 2.8).
- GPU — graphics processing unit.
- GRU — gated recurrent unit RNN architecture (Equation 2.39).
- HF — hazard function (Equation 2.7).
- HP — Hawkes process, a.k.a. self-exciting process (Equation 2.16).
- HPP — homogeneous Poisson process (Equation 2.12).
- IPP — inhomogeneous Poisson process (Equation 2.14).
- LL — log-likelihood (Equation 2.23).
- LSTM — long-short term memory RNN architecture.
- MLE — maximum likelihood estimation (Equation 2.25).
- MLP — multilayer preceptron (Equation 2.36).
- NLL — negative log-likelihood (Equation 2.23).
- PDF — probability density function (Equation 2.3).
- RNN — recurrent neural network.
- RP — renewal process (Equation 2.15).
- SCP — self-correcting process (Equation 2.19).
- SF — survival function (Equation 2.6).
- SPP — standard Poisson process, a.k.a. HPP with unit intensity (Equation 2.13).
- TPP — temporal point process.

Chapter 3

- FullyNN — Fully Neural Network Point Process.
- DSFlow — Deep sigmoidal flow.
- LogNormMix — TPP model based on the log-normal mixture distribution.
- RMTPP — Recurrent Marked Temporal Point Process model.
- SOSFlow — Sum-of-squares polynomial flow.

Chapter 4

- MRP — modulated renewal process (Equation 4.6).
- MMD — maximum mean discrepancy (Equation E.2).
- TriTPP — TPP model based on compisition of invertible maps (Figure 4.2).

Chapter 5

- ELBO — evidence lower bound.
- EM — expectation-maximization [16].
- MJP – Markov jump process.
- MMPP — Markov modulated Poisson process.
- MC — Monte Carlo.
- MCMC — Markov chain Monte Carlo.
- VI — variational inference.

Chapter 6

- 3S — sum-of-squared-spacings statistic.
- eCDF — empirical cumulative distribution function.
- GoF — goodness-of-fit.
- ID — in-distribution.
- KS — Kolmogorov–Smirnov.
- OoD — out-of-distribution.
- ROC AUC — area under the receiver operating characteristic curve.

C Characterizing a temporal point process

As mentioned in Section 2.2, the distribution of a TPP can be equivalently described using different functions. Here we provide a set of identities that allow us to switch between the different representations.

We can characterize the conditional distribution $P_i(t_i|t_1, \dots, t_{i-1})$ of the next arrival time given the history with one of the following functions:

- probability density function (PDF)

$$\begin{aligned}
p_i(t|t_1, \dots, t_{i-1}) &= \frac{d}{dt} F_i(t|t_1, \dots, t_{i-1}) && \text{(C.1)}\\
&= -\frac{d}{dt} S_i(t|t_1, \dots, t_{i-1}) && \text{(C.2)}\\
&= \phi_i(t|t_1, \dots, t_{i-1}) \exp\left(-\int_{t_{i-1}}^{t} \phi_i(u|t_1, \dots, t_{i-1}) du\right) && \text{(C.3)}\\
&= \frac{d}{dt} \Phi_i(t|t_1, \dots, t_{i-1}) \exp\left(-\Phi_i(t|t_1, \dots, t_{i-1})\right) && \text{(C.4)}
\end{aligned}$$

- cumulative distribution function (CDF)

$$\begin{aligned}
F_i(t|t_1, \dots, t_{i-1}) &= \int_{t_{i-1}}^{t} p_i(u|t_1, \dots, t_{i-1}) du && \text{(C.5)}\\
&= 1 - S_i(t|t_1, \dots, t_{i-1}) && \text{(C.6)}\\
&= 1 - \exp\left(-\int_{t_{i-1}}^{t} \phi_i(u|t_1, \dots, t_{i-1}) du\right) && \text{(C.7)}\\
&= 1 - \exp\left(-\Phi_i(t|t_1, \dots, t_{i-1})\right) && \text{(C.8)}
\end{aligned}$$

- survival function (SF)

$$\begin{aligned}
S_i(t|t_1, \dots, t_{i-1}) &= \int_{t}^{\infty} p_i(u|t_1, \dots, t_{i-1}) du && \text{(C.9)}\\
&= 1 - F_i(t|t_1, \dots, t_{i-1}) && \text{(C.10)}\\
&= \exp\left(-\int_{t_{i-1}}^{t} \phi_i(u|t_1, \dots, t_{i-1}) du\right) && \text{(C.11)}\\
&= \exp\left(-\Phi_i(t|t_1, \dots, t_{i-1})\right) && \text{(C.12)}
\end{aligned}$$

- hazard function (HF)

$$\begin{aligned}
\phi_i(t|t_1,\dots,t_{i-1}) &= \frac{p_i(t|t_1,\dots,t_{i-1})}{\int_t^\infty p_i(u|t_1,\dots,t_{i-1})du} && \text{(C.13)}\\
&= -\frac{d}{dt}\log(1-F_i(t|t_1,\dots,t_{i-1})) && \text{(C.14)}\\
&= -\frac{d}{dt}\log S_i(t|t_1,\dots,t_{i-1}) && \text{(C.15)}\\
&= \frac{d}{dt}\Phi_i(t|t_1,\dots,t_{i-1}) && \text{(C.16)}
\end{aligned}$$

- cumulative hazard function (CHF)

$$\begin{aligned}
\Phi_i(t|t_1,\dots,t_{i-1}) &= -\log\left(\int_t^\infty p_i(u|t_1,\dots,t_{i-1})du\right) && \text{(C.17)}\\
&= -\log(1-F_i(t|t_1,\dots,t_{i-1})) && \text{(C.18)}\\
&= -\log S_i(t|t_1,\dots,t_{i-1}) && \text{(C.19)}\\
&= \int_{t_{i-1}}^t \phi_i(u|t_1,\dots,t_{i-1})du && \text{(C.20)}
\end{aligned}$$

We can also use any of the above functions to compute the conditional intensity $\lambda^*(t)$ or the compensator $\Lambda^*(t)$. Suppose that t_{i-1} is the last event that occured before time t. Then we can use the following identities

$$\lambda^*(t) = \phi_i(t|t_1,\dots,t_{i-1}) \quad \text{(C.21)}$$

$$\Lambda^*(t) = \Phi_i(t|t_1,\dots,t_{i-1}) + \sum_{j=1}^{i-1}\Phi_j(t_j|t_1,\dots,t_{j-1}) \quad \text{(C.22)}$$

$$\Phi_i(t|t_1,\dots,t_{i-1}) = \Lambda^*(t) - \Lambda^*(t_{i-1}). \quad \text{(C.23)}$$

Combining these identities with Equations C.1–C.20 allows us to move between the conditional intensity and autoregressive characterizations of a TPP.

D Supplementary materials for Chapter 3

D.1 Survival and intensity functions for the proposed models

The survival function (SF) of a **normalizing flow** model can be obtained as follows. If u has a SF $Q(u)$ and $\tau = g(u)$ for increasing g, then the SF $S(\tau)$ of τ is obtained as

$$S(\tau) = Q(g^{-1}(\tau))$$

Since for both SOSFlow and DSFlow we can evaluate g^{-1} in closed form, $S(\tau)$ is easy to compute.

For the **log-normal mixture model**, SF is by definition equal to

$$S(\tau) = \sum_{k=1}^{K} w_k \bar{\Phi}\left(\frac{\log \tau - \mu_k}{s_k}\right)$$

where $\bar{\Phi}(\cdot)$ is the SF of a standard normal distribution.

Given the conditional PDF and SF, we can compute the conditional intensity $\lambda^*(t)$ for each model as

$$\lambda^*(t) = \frac{p_i^*(t - t_{i-1})}{S_i^*(t - t_{i-1})}$$

where t_{i-1} is the arrival time of most recent event before t [152]. This connection means that we are not losing any benefits of the intensity parametrization by directly modeling the inter-event times, as both ways to describe the model are in fact equivalent.

D.2 Discussion of constant & exponential intensity models

Constant intensity model as exponential distribution. The conditional intensity function of the constant intensity model [183] is defined as $\lambda^*(t_i) = \exp(\boldsymbol{v}^T \boldsymbol{h}_i + b)$, where $\boldsymbol{h}_i \in \mathbb{R}^H$ is the history embedding produced by an RNN, and $b \in \mathbb{R}$ is a learnable parameter. By setting $c = \exp(\boldsymbol{v}^T \boldsymbol{h}_i + b)$, it is easy to see that the PDF of the constant intensity model $p_i^*(\tau) = c \exp(-c\tau)$ corresponds to the exponential distribution.

Exponential intensity model as Gompertz distribution. PDF of a Gompertz distribution [193] is defined as

$$p(\tau|\alpha, \beta) = \alpha \exp\left(\beta\tau - \frac{\alpha}{\beta}\exp(\beta\tau) + \frac{\alpha}{\beta}\right)$$

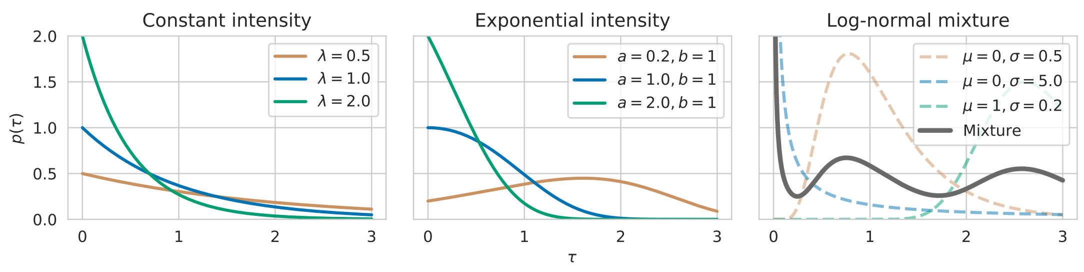


Figure D.1: Different choices for modeling $p(\tau)$: exponential distribution (left), Gompertz distribution (center), log-normal mixture (right). Mixture distribution can approximate any density while being tractable and easy to sample from.

for $\alpha, \beta > 0$. The two parameters α and β define its shape and rate, respectively. For any choice of its parameters, Gompertz distribution is unimodal and light-tailed. The mean of the Gompertz distribution can be computed as $\mathbb{E}[\tau] = \frac{1}{\beta} \exp\left(\frac{\alpha}{\beta}\right) \operatorname{Ei}(-\frac{\alpha}{\beta})$, where $\operatorname{Ei}(z) = \int_{-z}^{\infty} \exp(-v)/v \, dv$ is the exponential integral function (that can be approximated numerically).

The conditional intensity function of the exponential intensity model [52] is defined as $\lambda^*(t_i) = \exp(w(t_i - t_{i-1}) + \boldsymbol{v}^T \boldsymbol{h}_i + b)$, where $\boldsymbol{h}_i \in \mathbb{R}^H$ is the history embedding produced by an RNN, and $\boldsymbol{v} \in \mathbb{R}^H, b \in \mathbb{R}, w \in \mathbb{R}_+$ are learnable parameters. By defining $d = \boldsymbol{v}^T \boldsymbol{h}_i + b$, we obtain the PDF of the exponential intensity model [52, Equation 12]

$$p_i^*(\tau | w, d) = \exp\left(w\tau + d - \frac{1}{w}\exp(w\tau + d) + \frac{1}{w}\exp(d)\right).$$

By setting $\alpha = \exp(d)$ and $\beta = w$ we see that the exponential intensity model is equivalent to a Gompertz distribution.

Discussion. Figure D.1 shows densities that can be represented by exponential and Gompertz distributions. Even though the history embedding $\boldsymbol{h}_i$ produced by an RNN may capture rich information, the resulting distribution $p_i^*(\tau)$ for both models has very limited flexibility, is unimodal and light-tailed. In contrast, a flow-based or a mixture model is significantly more flexible and can approximate any density.

D.3 Discussion of the FullyNN model

Summary. The main idea of the approach by [139] is to model the cumulative hazard function (CHF, Equation 2.8) of the inter-event times

$$\Phi_i^*(\tau) = \int_0^{\tau} \lambda^*(t_{i-1} + s) ds$$

using a feedforward neural network with non-negative weights

$$\Phi_i^*(\tau) := f_i(\tau) = \operatorname{softplus}(\boldsymbol{W}^{(3)} \tanh(\boldsymbol{W}^{(2)} \tanh(\boldsymbol{W}^{(1)} \tau + \tilde{\boldsymbol{b}}_i^{(1)}) + \boldsymbol{b}^{(2)}) + \boldsymbol{b}^{(3)}) \quad \text{(D.1)}$$

where $\tilde{\boldsymbol{b}}_i^{(1)} = \boldsymbol{V}\boldsymbol{h}_i + \boldsymbol{b}^{(0)}$, $\boldsymbol{h}_i \in \mathbb{R}^H$ is the history embedding, $\boldsymbol{W}^{(1)} \in \mathbb{R}_+^{D\times 1}$, $\boldsymbol{W}^{(2)} \in \mathbb{R}_+^{D\times D}$, $\boldsymbol{W}^{(3)} \in \mathbb{R}_+^{1\times D}$ are non-negative weight matrices, and $\boldsymbol{V} \in \mathbb{R}^{D\times H}$, $\boldsymbol{b}^{(0)} \in \mathbb{R}^D$, $\boldsymbol{b}^{(2)} \in \mathbb{R}^D$, $\boldsymbol{b}^{(3)} \in \mathbb{R}$ are the remaining model parameters.

FullyNN as a normalizing flow. Let $u \sim \text{Exponential}(1)$, that is

$$F(u) = 1 - \exp(-u) \qquad\qquad p(u) = \exp(-u)$$

We can view $f : \mathbb{R}_+ \to \mathbb{R}_+$ as a transformation that maps τ to u

$$u = f_i(\tau) \iff \tau = f_i^{-1}(u)$$

We can now use the change of variables formula to obtain the conditional CDF and PDF of τ.

Alternatively, we can obtain the conditional intensity as

$$\lambda^*(t) = \frac{d}{d\tau}\Phi_i^*(\tau) = \frac{d}{d\tau} f_i(\tau)$$

and use the fact that $\Phi_i^*(\tau) = \int_0^\tau \lambda^*(t_{i-1} + s)ds$.

Both approaches lead to the same conclusion

$$F_i^*(\tau) = 1 - \exp(-f_i(\tau)) \qquad\qquad p_i^*(\tau) = \exp(-f_i(\tau))\frac{d}{d\tau}f_i(\tau)$$

However, the first approach also provides intuition on how to draw samples $\tilde{\tau}$ from the resulting distribution $p_i^*(\tau)$

1. Sample $\tilde{u} \sim \text{Exponential}(1)$.

2. Obtain $\tilde{\tau}$ by solving $f(\tau) - \tilde{u} = 0$ for τ (using e.g. bisection method).

This recovers the inverse transform method (Algorithm 3). Similarly to other flow-based models, sampling from the FullyNN model cannot be done exactly and requires a numerical approximation since f cannot be inverted analytically.

An even more important shortcoming is that FullyNN does not define a valid TPP. The chosen parametrization of $\Phi_i^*(\tau)$ assigns a non-zero probability mass to the $(-\infty, 0)$ interval, which violates the assumption that inter-event times are strictly positive with probability 1. This assumption corresponds to the constraint $\text{Prob}(\tau_i \le 0) = F_i^*(0) = 0$, or equivalently $\Phi_i^*(0) = 0$. However, we can see that

$$\Phi_i^*(0) = f(0) = \text{softplus}(\boldsymbol{W}^{(3)}\tanh(\boldsymbol{W}^{(2)}\tanh(\tilde{\boldsymbol{b}}^{(1)}) + \boldsymbol{b}^{(2)}) + \boldsymbol{b}^{(3)}) > 0$$

which means that the FullyNN model assigns a non-zero probability to negative inter-event times, and therefore doesn't define a valid TPP.

D.4 Implementation details

D.4.1 Shared architecture

We implement SOSFlow, DSFlow and LogNormMix, together with baselines: RMTPP (Gompertz distribution), exponential distribution and a FullyNN model. All of them share the same pipeline, from the data preprocessing to the parameter tuning and model selection, differing only in the way we calculate $p^*(\tau)$. This way we ensure a fair evaluation. Our implementation uses Pytorch [145].

From arival times t_i we calculate the inter-event times $\tau_i = t_i - t_{i-1}$. Since they can contain very large values, RNN takes log-transformed and centered inter-event time and produces $\boldsymbol{h}_i \in \mathbb{R}^H$. In case we have marks, we additionally input m_i — the index of the mark class from which we get mark embedding vector m_i. In some experiments we use extra conditional information, such as metadata $\boldsymbol{y}_i$ and sequence embedding $\boldsymbol{e}_j$, where j is the index of the sequence.

As illustrated in Section 3.2.3 we generate the parameters $\boldsymbol{\varphi}$ of the distribution $p^*(\tau_i)$ from $[\boldsymbol{h}_i||\boldsymbol{y}_i||\boldsymbol{e}_j]$ using an affine layer. We apply a transformation of the parameters to enforce the constraints, if necessary.

All decoders are implemented using a common framework relying on normalizing flows. By defining the base distribution $q(u)$ and the inverse transformation $(g_1^{-1} \circ \dots \circ g_M^{-1})$ we can evaluate the PDF $p_i^*(\tau)$ at any τ, which allows us to train with maximum likelihood (Section 3.2.1).

D.4.2 Log-normal mixture

The log-normal mixture distribution is defined in Equation 3.8. We generate the parameters of the distribution $\boldsymbol{w} \in \mathbb{R}^K, \boldsymbol{\mu} \in \mathbb{R}^K, \boldsymbol{s} \in \mathbb{R}^K$ (subject to $\sum_k w_k = 1, w_k \geq 0$ and $s_k > 0$), using an affine transformation (3.10). The log-normal mixture is equivalent to the following normalizing flow model

$$\begin{aligned} u_1 &\sim \text{GaussianMixture}(\boldsymbol{w}, \boldsymbol{\mu}, \boldsymbol{s}) \\ u_2 &= a u_1 + b \\ \tau &= \exp(u_2) \end{aligned}$$

By using the affine transformation $u_2 = au_1 + b$ before the exp transformation, we obtain a better initialization, and thus faster convergence. This is similar to the batch normalization flow layer [50], except that $b = \frac{1}{N}\sum_{i=1}^N \log \tau_i$ and $a = \sqrt{\frac{1}{N}\sum_{i=1}^N (\log \tau_i - b)}$ are estimated using the entire training, not using batches.

Forward direction samples a value from a Gaussian mixture, applies an affine transformation and applies exp. In the *backward* direction we apply log-transformation to an observed data, center it with an affine layer and compute the density under the Gaussian mixture.

D.4.3 Baselines

We implement FullyNN model [139] as described in Appendix D.3, using the official implementation as a reference.[1] The model uses feed-forward neural network with non-negative weights (enforced by clipping values at 0 after every gradient step). Output of the network is a cumulative intensity function $\Phi_i^*(\tau)$ from which we can get intensity function $\lambda^*(t_{i-1} + \tau)$ as a derivative w.r.t. τ using automatic differentiation in Pytorch.

We implement RMTPP / Gompertz distribution [52][2] and the exponential distribution [183] models as described in Appendix D.2.

All of the above methods define the conditional PDF $p_i^*(\tau)$. Since the inter-event times may come at very different scales, we apply a linear scaling $\tilde{\tau} = a\tau$, where $a = \frac{1}{N}\sum_{i=1}^{N} \tau_i$ is estimated from the data. This ensures a good initialization for all models and speeds up training.

D.4.4 Deep sigmoidal flow

A single layer of DSFlow model is defined as

$$f_{\boldsymbol{\theta}}^{DSF}(x) = \sigma^{-1}\left(\sum_{k=1}^{K} w_k \sigma\left(\frac{x - \mu_k}{s_k}\right)\right)$$

with parameters $\boldsymbol{\theta} = \{\boldsymbol{w} \in \mathbb{R}^K, \boldsymbol{\mu} \in \mathbb{R}^K, \boldsymbol{s} \in \mathbb{R}^K\}$ (subject to $\sum_k w_k = 1, w_k \geq 0$ and $s_k > 0$). We obtain the parameters of each layer using Equation 3.10. We define $p(\tau)$ through the inverse transformation $(g_1^{-1} \circ \cdots \circ g_M^{-1})$, as described in Section 3.2.1.

$$\begin{aligned}
&u_M = g_M^{-1}(\tau) = \log \tau \\
&\cdots \\
&u_m = g_m^{-1}(u_{m+1}) = f_{\boldsymbol{\theta}_m}^{DSF}(u_{m+1}) \\
&\cdots \\
&u_1 = \sigma(u_2) \\
&u_1 \sim q_1(u_1) = \text{Uniform}([0, 1])
\end{aligned}$$

We use the batch normalization flow layer [50] between every pair of consecutive layers, which significantly speeds up convergence.

D.4.5 Sum-of-squares polynomial flow

A single layer of SOSFlow model is defined as

$$f^{SOS}(x) = a_0 + \sum_{k=1}^{K}\sum_{p=0}^{R}\sum_{q=0}^{R} \frac{a_{p,k}a_{q,k}}{p + q + 1} x^{p+q+1}$$

[1]`https://github.com/omitakahiro/NeuralNetworkPointProcess`

[2]`https://github.com/musically-ut/tf_rmtpp`

There are no constraints on the polynomial coefficients $\boldsymbol{a} \in \mathbb{R}^{(R+1)\times K}$. We obtain $\boldsymbol{a}$ similarly to Equation 3.10 as $\boldsymbol{a} = \boldsymbol{V_a c} + \boldsymbol{b_a}$, where $\boldsymbol{c}$ is the context vector.

We define $p(\tau)$ by through the inverse transformation $(g_1^{-1} \circ \cdots \circ g_M^{-1})$, as described in Section 3.2.1.

$$\begin{aligned}
&u_M = g_M^{-1}(\tau) = \log \tau \\
&\cdots \\
&u_m = g_m^{-1}(u_{m+1}) = f_{\boldsymbol{\theta}_m}^{SOS}(u_{m+1}) \\
&\cdots \\
&u_1 = \sigma(u_2) \\
&u_1 \sim q_1(u_1) = \text{Uniform}(0, 1)
\end{aligned}$$

Same as for DSFlow, we use the batch normalization flow layer between every pair of consecutive layers. When implementing SOSFlow, we used Pyro[3] for reference.

D.5 Dataset statistics

D.5.1 Synthetic data

Synthetic data is generated according to Omi et al. [139] using simple conventional TPPs. We sample 64 sequences for each process, each sequence containing 1024 events.

Poisson (Equation 2.13). Conditional intensity function for a homogeneous (or stationary) Poisson point process is given as $\lambda^*(t) = 1$. Constant intensity corresponds to the exponential inter-event time distribution.

Renewal (Equation 2.15). A stationary process defined by a log-normal probability density function $p(\tau)$, where we set the parameters to be $\mu = 1.0$ and $\sigma = 6.0$. Sequences appear clustered.

Self-correcting (Equation 2.19). Unlike the previous two, this point process depends on the history and is defined by a conditional intensity function $\lambda^*(t) = \exp(t - \sum_{t_i < t} 1)$. After every new event the intensity drops, inhibiting the future activity. The resulting point patterns appear regular.

Hawkes. We use a self-exciting point process with a conditional intensity function

$$\lambda^*(t) = \mu + \sum_{t_i < t} \sum_{j=1}^{M} \alpha_j \beta_j \exp(-\beta_j (t - t_i)).$$

As per Omi et al. [139], we create two different datasets:

- **Hawkes1** with $M = 1$, $\mu = 0.02$, $\alpha_1 = 0.8$ and $\beta_1 = 1.0$.
- **Hawkes2** with $M = 2$, $\mu = 0.2$, $\alpha_1 = 0.4$, $\beta_1 = 1.0$, $\alpha_2 = 0.4$ and $\beta_2 = 20$.

For the imputation experiment we use Hawkes1 to generate the data and remove some of the events.

[3] `https://pyro.ai/` [14]

Table D.1: Dataset statistics.

Dataset name	Number of sequences	Number of events
LastFM	929	1268385
Reddit	10000	672350
Stack Overflow	6633	480414
MOOC	7047	396633
Wikipedia	1000	157471
Yelp	300	215146

D.5.2 Real-world data

In addition we use real-world datasets that are described bellow. Table D.1 shows their summary. All datasets have a large amount of unique sequences and the number of events per sequence varies a lot. Using marked temporal point processes to predict the type of an event is feasible for some datasets (e.g. when the number of classes is low), and is meaningless for other.

LastFM [25]. The dataset contains sequences of songs that selected users listen over time. Artists are used as an event type.

Reddit [102]. On this social network website users submit posts to subreddits. In the dataset, most active subreddits are selected, and posts from the most active users on those subreddits are recodered. Each sequence corresponds to a list of submissions a user makes. The data contains 984 unique subreddits that we use as classes in mark prediction.

Stack Overflow [52]. Users of a question-answering website get rewards (called badges) over time. A sequence contains a list of rewards for each user. Only the most active users are selected and only those badges that users can get more than once.

MOOC [102]. Contains the interaction of students with an online course system. An interaction is an event and can be of various types (97 unique types), e.g. watching a video, solving a quiz etc.

Wikipedia.[102]. A sequence corresponds to edits of a Wikipedia page. The dataset contains most edited pages and users that have an activity (number of edits) above a certain threshold.

Yelp.[4] We use the data from the review forum and consider the reviews for the 300 most visited restaurants in Toronto. Each restaurant then has a corresponding sequence of reviews over time.

[4]`https://www.yelp.com/dataset/challenge`

D.6 Additional discussion of the experiments

D.6.1 Event time prediction using history

Detailed setup. Each dataset consists of multiple sequences of inter-event times. We consider 10 train/validation/test splits of the sequences (of sizes 60%/20%/20%). We train all model parameters by minimizing the negative log-likelihood (NLL) of the training sequences. After splitting the data into the 3 sets, we break down long training sequences into sequences of length at most 128. Optimization is performed using Adam [98] with learning rate 10^{-3}. We perform training using mini-batches of 64 sequences. We train for up to 2000 epochs (1 epoch = 1 full pass through all the training sequences). For all models, we compute the validation loss at every epoch. If there is no improvement for 100 epochs, we stop optimization and revert to the model parameters with the lowest validation loss.

We select hyperparameter configuration for each model that achieves the lowest average loss on the validation set. For each model, we consider different values of L_2 regularization strength $\lambda_{\text{reg}} \in \{0, 10^{-5}, 10^{-3}\}$. Additionally, for SOSFlow we tune the number of transformation layers $M \in \{1, 2, 3\}$ and for DSFlow $M \in \{1, 2, 3, 5, 10\}$. We have chosen the values of K such that the mixture model has approximately the same number of parameters as a 1-layer DSFlow or a 1-layer FullyNN model. More specifically, we set $K = 64$ for LogNormMix, DSFlow and FullyNN. We found all these models to be rather robust to the choice of K, as can be seen in Table D.2 for LogNormMix. For SOSFlow we used $K = 4$ and $R = 3$, resulting in a polynomial of degree 7 (per each layer). Higher values of R led to unstable training, even when using batch normalization.

Additional discussion. In this experiment, we only condition the distribution $p_i^*(\tau)$ on the history embedding $\boldsymbol{h}_i$. We do not learn sequence embeddings $\boldsymbol{e}_j$ since they can only be learned for the training sequences, and not fore the validation/test sets.

There are two important aspects related to the NLL loss values that we report. First, the absolute loss values can be arbitrarily shifted by rescaling the data. Assume, that we have a distribution $p(\tau)$ that models the distribution of τ. Now assume that we are interested in the distribution $q(x)$ of $x = a\tau$ (for $a > 0$). Using the change of variables formula, we obtain $\log q(x) = \log p(\tau) + \log a$. This means that by scaling the data we can arbitrarily offset the log-likelihood score that we obtain. Therefore, the absolute values of of the (negative) log-likelihood $\mathcal{L}$ for different models are of little interest — all that matters are the differences between them.

The loss values are dependent on the train/val/test split. Assume that model 1 achieves loss values $\mathcal{L}_1 = \{1.0, 3.0\}$ on two train/val/test splits, and model 2 achieves $\mathcal{L}_2 = \{2.0, 4.0\}$ on the same splits. If we first aggregate the scores and report the average $\hat{\mathcal{L}}_1 = 2.0 \pm 1.0$, $\hat{\mathcal{L}}_2 = 3.0 \pm 1.0$, it may seem that the difference between the two models is not significant. However, if we first compute the differences and then aggregate $(\mathcal{L}_2 - \mathcal{L}_1) = 1.0 \pm 0.0$ we see a different picture. Therefore, we use the latter strategy in Figure 3.2. For completeness, we also report the numbers obtained using the first strategy in Table D.3.

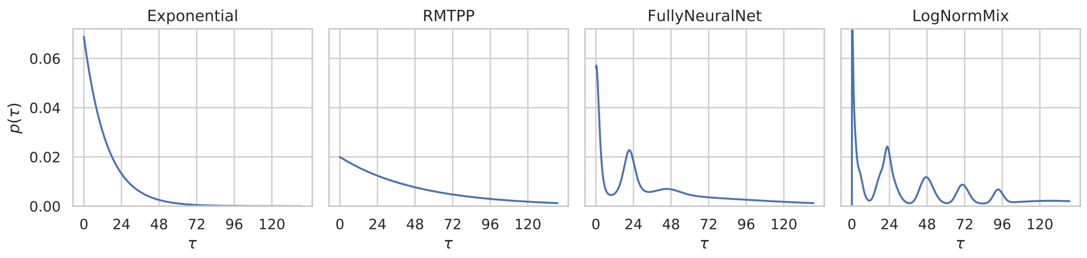


Figure D.2: Models learn different conditional distribution $p^*(\tau)$ on Yelp dataset. Since check-ins occur during the opening hours, true distribution of the next check-in resembles the one on the right.

As a baseline, we also considered the constant intensity / exponential distribution model [183]. However, we excluded the results for it from Figure 3.2, since it consistently achieved the worst loss values and had high variance. We still include the results for the constant intensity model in Table D.3. We also performed all the experiments on the synthetic datasets (Appendix D.5.1). The results are shown in Table D.4, together with NLL scores under the true model. We see that LogNormMix and DSFlow, besides achieving the best results, recover the true distribution.

Finally, in Figure D.2 we plot the conditional distribution $p^*(\tau)$ with models trained on Yelp dataset. The events represent check-ins into a specific restaurant. Since check-ins mostly happen during the opening hours, the inter-event time is likely to be on the same day (0h), next day (24h), the day after (48h), etc. LogNormMix can fully recover this behavior from data while others either cannot learn multimodal distributions (e.g. RMTPP) or struggle to capture it (e.g. FullyNN).

Table D.2: Time NLL of the LogNormMix model for different numbers K of mixture components.

K	2	4	8	16	32	64
Reddit	10.239	10.208	10.189	10.185	10.191	10.192
LastFM	-2.828	-2.879	-2.881	-2.880	-2.877	-2.860
MOOC	6.246	6.053	6.055	6.055	6.050	5.660
Stack Overflow	14.461	14.438	14.435	14.435	14.436	14.428
Wikipedia	8.399	8.389	8.385	8.384	8.384	8.386
Yelp	13.169	13.103	13.058	13.045	13.032	13.024
Poisson	1.006	0.992	0.991	0.991	0.990	0.991
Renewal	0.256	0.254	0.254	0.254	0.256	0.259
Self-correcting	0.831	0.785	0.782	0.783	0.784	0.784
Hawkes1	0.530	0.523	0.532	0.532	0.523	0.523
Hawkes2	0.036	0.026	0.024	0.024	0.026	0.024

Table D.3: Time prediction test NLL on real-world data.

	Reddit	LastFM	MOOC	Stack Overflow	Wikipedia	Yelp
LogNormMix	**10.19 ± 0.078**	**-2.88 ± 0.147**	**6.03 ± 0.092**	**14.44 ± 0.013**	**8.39 ± 0.079**	**13.02 ± 0.070**
DSFlow	10.20 ± 0.074	**-2.88 ± 0.148**	**6.03 ± 0.090**	**14.44 ± 0.019**	8.40 ± 0.090	13.09 ± 0.065
SOSFlow	10.27 ± 0.106	-2.56 ± 0.133	6.27 ± 0.058	14.47 ± 0.049	8.44 ± 0.120	13.21 ± 0.068
FullyNN	10.23 ± 0.072	-2.84 ± 0.179	6.83 ± 0.152	14.45 ± 0.014	8.40 ± 0.086	13.04 ± 0.073
LogNormal	10.38 ± 0.077	-2.60 ± 0.140	6.53 ± 0.016	14.62 ± 0.013	8.52 ± 0.078	13.44 ± 0.074
RMTPP	10.88 ± 0.293	-1.30 ± 0.164	10.65 ± 0.023	14.51 ± 0.014	10.02 ± 0.085	13.36 ± 0.056
Exponential	11.07 ± 0.070	-1.28 ± 0.152	10.64 ± 0.026	18.48 ± 3.257	10.03 ± 0.083	13.78 ± 1.250

Table D.4: Time prediction test NLL on synthetic data.

	Poisson	Renewal	Self-correcting	Hawkes1	Hawkes2
True model	0.999	0.254	0.757	0.453	-0.043
LogNormMix	**0.99 ± 0.006**	**0.25 ± 0.010**	**0.78 ± 0.003**	**0.52 ± 0.047**	**0.02 ± 0.049**
DSFlow	**0.99 ± 0.006**	**0.25 ± 0.010**	**0.78 ± 0.002**	**0.52 ± 0.047**	**0.02 ± 0.050**
SOSFlow	1.00 ± 0.013	**0.25 ± 0.010**	0.88 ± 0.011	0.59 ± 0.056	0.06 ± 0.046
FullyNN	1.00 ± 0.006	0.28 ± 0.013	**0.78 ± 0.004**	0.55 ± 0.047	0.06 ± 0.047
LogNormal	1.08 ± 0.008	**0.25 ± 0.010**	1.03 ± 0.006	0.55 ± 0.047	0.06 ± 0.049
RMTPP	**0.99 ± 0.006**	1.01 ± 0.023	**0.78 ± 0.003**	0.74 ± 0.057	0.69 ± 0.058
Exponential	**0.99 ± 0.006**	1.00 ± 0.023	0.94 ± 0.002	0.74 ± 0.055	0.69 ± 0.054

D.6.2 Learning with marks

Detailed setup. We use the same setup as in Appendix D.6.1, except two differences. For learning in a marked temporal point process, we mimic the architecture from Du et al. [52]. The RNN takes a tuple (τ_i, m_i) as input at each time step, where m_i is the mark. Moreover, the loss function now includes a term for predicting the next mark: $\mathcal{L}_{\text{total}} = -\sum_{i=1}^{N} [\log p_i^*(\tau_i) + \log p_i^*(m_i)]$.

The next mark m_i at time t_i is predicted using a categorical distribution $p^*(m_i)$. The distribution is parametrized by the vector $\boldsymbol{\pi}_i$, where $\pi_{i,c}$ is the probability of event $m_i = c$. We obtain $\boldsymbol{\pi}_i$ using the history embedding $\boldsymbol{h}_i$ passed through a feedforward neural network

$$\boldsymbol{\pi}_i = \text{softmax}\left(\boldsymbol{V}_{\boldsymbol{\pi}}^{(2)} \tanh(\boldsymbol{V}_{\boldsymbol{\pi}}^{(1)} \boldsymbol{h}_i + \boldsymbol{b}_{\boldsymbol{\pi}}^{(1)}) + \boldsymbol{b}_{\boldsymbol{\pi}}^{(2)}\right)$$

where $\boldsymbol{V}_{\boldsymbol{\pi}}^{(1)}, \boldsymbol{V}_{\boldsymbol{\pi}}^{(2)} \boldsymbol{b}_{\boldsymbol{\pi}}^{(1)}, \boldsymbol{b}_{\boldsymbol{\pi}}^{(2)}$ are the parameters of the neural network.

Additional discussion. In Figure 3.2 (right) we reported the differences in time NLL between different models $\mathcal{L}_{\text{time}} = -\sum_{i=1}^{N} \log p_i^*(\tau_i)$. In Table D.5 we additionally provide the total NLL $\mathcal{L}_{\text{total}} = -\sum_{i=1}^{N} [\log p_i^*(\tau_i) + \log p_i^*(m_i)]$ averaged over multiple splits.

Using marks as input to the RNN improves time prediction quality for all the models. However, since we assume that the marks are conditionally independent of the time given the history (as was done in earlier works), all models have similar mark prediction accuracy.

Table D.5: Time and total NLL and mark accuracy when learning a marked TPP.

	Time NLL		Total NLL		Mark accuracy	
	Reddit	MOOC	Reddit	MOOC	Reddit	MOOC
LogNormMix	**10.28 ± 0.066**	**5.75 ± 0.040**	12.40 ± 0.094	7.58 ± 0.047	0.62±0.014	0.45±0.003
DSFlow	**10.28 ± 0.073**	5.78 ± 0.067	**12.39 ± 0.064**	**7.52 ± 0.074**	0.62±0.013	0.45±0.004
SOSFlow	10.35 ± 0.106	6.06 ± 0.084	12.49 ± 0.158	7.78 ± 0.107	0.62±0.013	**0.46±0.009**
FullyNN	10.41 ± 0.079	6.22 ± 0.224	12.51 ± 0.094	7.93 ± 0.230	**0.63±0.013**	**0.46±0.004**
LogNormal	10.42 ± 0.076	6.38 ± 0.019	12.51 ± 0.080	8.11 ± 0.026	0.62±0.013	0.42±0.005
RMTPP	11.15 ± 0.061	10.29 ± 0.209	13.26 ± 0.085	12.14 ± 0.220	0.62±0.014	0.41±0.006

D.6.3 Learning with additional conditional information

Detailed setup. In the Yelp dataset, the task is to predict the time τ_i until the next customer check-in, given the history of check-ins up until the current time t_{i-1}. We want to verify our intuition that the distribution $p_i^*(\tau)$ depends on the current time t_{i-1}. For example, $p_i^*(\tau)$ might be different depending on whether it is a weekday and / or it is an evening hour. Unfortunately, a model that processes the history with an RNN cannot easily obtain this information. Therefore, we provide this information directly as a context vector $\boldsymbol{y}_i$ when modeling $p_i^*(\tau)$.

The first entry of context vector $\boldsymbol{y}_i \in \{0, 1\}^2$ indicates whether the previous event t_{i-1} took place on a weekday or a weekend, and the second entry indicates whether t_{i-1} was in the 5PM–11PM time window. To each of the four possibilities we assign a learnable 64-dimensional embedding vector. The distribution of $p_i^*(\tau)$ until the next event depends on the embedding vector of the time stamp t_{i-1} of the most recent event.

D.6.4 Missing data imputation

Detailed setup. The dataset for the experiment is generated as a two step process: 1) We generate a sequence of 100 events from the model used for Hawkes1 dataset (Appendix D.5.1) resulting in a sequence of arrival times $\{t_1, \dots t_N\}$, 2) We choose random t_i and remove all the events that fall inside the interval $[t_i, t_{i+k}]$ where k is selected such that the interval length is approximately $t_N/3$.

We consider three strategies for learning with missing data (shown in Figure 3.5 (left)):

a) **No imputation**. The missing block spans the time interval $[t_i, t_{i+k}]$. We simply ignore the missing data, i.e. training objective $\mathcal{L}_{\text{time}}$ will include an inter-event time $\tau = t_{i+k} - t_i$.

b) **Mean imputation**. We estimate the average inter-event time $\hat{\tau}$ from the observed data, and impute events at times $\{t_i + n\hat{\tau} \text{ for } n \in \mathbb{N}, \text{ such that } t_i + n\hat{\tau} < t_{i+k}\}$. These imputed events are fed into the history-encoding RNN, but are not part of the training objective.

c) **Sampling** . The RNN encodes the history up to and including t_i and produces $\boldsymbol{h}_i$ that we use to define the distribution $p^*(\tau|\boldsymbol{h}_i)$. We draw a sample $\tau_j^{(\text{imp})}$ form this distribution and feed it into the RNN. We keep repeating this procedure until the samples get past the point t_{i+k}. The imputed inter-event times $\tau_j^{(\text{imp})}$ are affecting the hidden state of the RNN (thus influencing the likelihood of future observed inter-event times $\tau_i^{(\text{obs})}$).

We sample multiple such sequences in order to approximate the *expected* NLL of the observed inter-event times $\mathbb{E}_{\tau^{(\text{imp})} \sim p^*}\left[-\sum_i \log p^*(\tau_i^{(\text{obs})})\right]$. Since this objective includes an expectation that depends on p^*, we make use of reparametrization sampling to obtain the gradients w.r.t. the distribution parameters [121].

D.6.5 Sequence embedding

Detailed setup. When learning sequence embeddings, we train the model as described in Appendix D.6.1, besides one difference. First, we pre-train the sequence embeddings $\boldsymbol{e}_j$ by disabling the history embedding $\boldsymbol{h}_i$ and optimizing $-\sum_i \log p_i(\tau_i | \boldsymbol{e}_j)$. Afterwards, we enable the history and minimize $-\sum_i \log p_i(\tau_i | \boldsymbol{e}_j, \boldsymbol{h}_i)$.

In Figure 3.6 the top row shows samples generated using $\boldsymbol{e}_{SCP}$, embedding of a self-correcting sequence, the bottom row was generated using $\boldsymbol{e}_{RP}$, embedding of a renewal sequence, and the middle row was generated using $1/2(\boldsymbol{e}_{SCP} + \boldsymbol{e}_{RP})$, an average of the two embeddings.

E Supplementary materials for Chapter 4

E.1 Implementation details

E.1.1 Batch processing

By representing TPP densities with transformations, we can implement both density evaluation and sampling efficiently and in parallel. Our implementation enables parallelism not only for the events t_i of a single sequence, but also for entire batches consisting of multiple sequences of different length.

First, consider a single event sequence $\mathcal{T} = (1, 2.5, 4)$ with $N = 3$ events that was observed on the time interval $[0, 5]$. We represent this sequence by a vector

$$\boldsymbol{t} = \begin{bmatrix} 1 & 2.5 & 4 & 5 \end{bmatrix}.$$

The vector $\boldsymbol{t}$ contains timestamps of the events, and the last entry corresponds to $T = 5$, the length of the observed interval. We additionally introduce a binary mask $\boldsymbol{b}$ that tells us which entries of the padded vector $\boldsymbol{t}$ correspond to actual events (i.e., not padding)

$$\boldsymbol{b} = \begin{bmatrix} 1 & 1 & 1 & 0 \end{bmatrix}.$$

We implement the transformation $\boldsymbol{F}$ similarly to normalizing flow frameworks like `pyro` [14] and `torch.distributions` [145]. We define a method `forward` that computes $\boldsymbol{z}$, the result of the transformation, and $\boldsymbol{j}$, logarithm of the diagonal entries of the Jacobian $J_{\boldsymbol{F}}(\boldsymbol{t})$:

$$\boldsymbol{z} = \boldsymbol{F}(\boldsymbol{t}) = \begin{bmatrix} z_1 & z_2 & z_3 & z_4 \end{bmatrix} \qquad \boldsymbol{j} = \begin{bmatrix} \log\left|\frac{\partial z_1}{\partial t_1}\right| & \log\left|\frac{\partial z_2}{\partial t_2}\right| & \log\left|\frac{\partial z_3}{\partial t_3}\right| & \log\left|\frac{\partial z_4}{\partial t_4}\right| \end{bmatrix}$$

From the definition of $\boldsymbol{F}$ (Section 4.2), we can see that the last entry of $\boldsymbol{z}$ (that we denote as z_{-1}) corresponds to $\Lambda^*(T)$. Also, each entry j_i of $\boldsymbol{j}$ corresponds to $\log\left|\frac{\partial \Lambda^*(t_i)}{\partial t_i}\right|$. Therefore, we can compute the log-density $\log p(\boldsymbol{t})$ as

$$\begin{aligned} \log p(\boldsymbol{t}) &= \texttt{sum}(\boldsymbol{b} \odot \boldsymbol{j}) - z_{-1} \\ &= \sum_{i=1}^{N'} m_i \log\left|\frac{\partial z_i}{\partial t_i}\right| - z_{-1} \\ &= \sum_{i=1}^{N} \log\left|\frac{\partial \Lambda^*(t_i)}{\partial t_i}\right| - \Lambda^*(T) \end{aligned} \tag{E.1}$$

where N' denotes the length *with* the padding. We can verify that this is equal to the logarithm of the TPP density in Equation 4.1. Note that if we use a longer padding, such as

$$\boldsymbol{t} = \begin{bmatrix}1 & 2.5 & 4 & 5 & 5 & 5 & 5\end{bmatrix} \qquad \boldsymbol{b} = \begin{bmatrix}1 & 1 & 1 & 0 & 0 & 0 & 0\end{bmatrix}$$

then Equation E.1 will still correctly compute the log-likelihood for the sequence. This observation allows to process multiple sequences $\{\boldsymbol{t}^{(1)}, \boldsymbol{t}^{(2)}, ...\}$ in a single batch. We simply pad all the sequences with T up to the length of the longest sequence, stack them into a matrix of shape `[batch_size, max_seq_len]` and process all of them in parallel.

As described in Section 4.2.3, we actually define $\boldsymbol{F}$ by stacking multiple transformations. We sequentially call the `forward` method for each transformation in the chain to obtain the final $\boldsymbol{z}$, and sum up the log-diagonals of the Jacobians $\boldsymbol{j}$ along the way. Each transformation and its Jacobian can be evaluated in parallel in linear time, making the whole operation efficient.

E.1.2 Sampling

Sampling is implemented similarly. We start by simulating a vector $\tilde{\boldsymbol{z}}$ from the standard Poisson process (Algorithm 1). The length of $\tilde{\boldsymbol{z}}$ must be "long enough" (more on this later). We define the method `inverse` that computes $\tilde{\boldsymbol{t}} = \boldsymbol{F}^{-1}(\tilde{\boldsymbol{z}})$. We obtain a final sample $\boldsymbol{t}$ by clipping the entries of $\tilde{\boldsymbol{t}}$ as $t_i = \min\{\tilde{t}_i, T\}$. If we would like to compute the density of the generated sample $\boldsymbol{t}$, we will also need the mask $\boldsymbol{b}$ that can be obtained as $b_i = \mathbb{1}(\tilde{t}_i < T)$. In some use cases, such as entropy maximization or variational inference, we need to use a differentiable approximation to the mask $b_i = \sigma_\zeta(T - \tilde{t}_i)$. This recovers our relaxation from Chapter 5.

By slightly abusing the notation, we use N' to denote the number of events in our initial HPP sample $\tilde{\boldsymbol{z}} = (\tilde{z}_1, ..., \tilde{z}_{N'})$. N' must be large enough, such that the event $\tilde{t}_{N'}$ (corresponding to $\tilde{z}'_N$) happens after T. We can easily ensure this by setting N' to some large number (e.g., 100 or 1000), and increasing it if for some sample $\tilde{t}_{N'}$ is less than T. As we saw in Figure 4.3, using larger sequence length leads to no noticeable computational overhead when using GPU.

E.1.3 Ensuring that the TPP is valid

We showed in Section 4.2 that every TPP density $p(\mathcal{T})$ corresponds to a differentiable increasing triangular map $\boldsymbol{F}$ defined by the compensator Λ^*. When directly parametrizing $\boldsymbol{F}$, we need to check one of the two equivalent conditions to ensure that our map $\boldsymbol{F}$ defines a valid temporal point process.

Condition 1. The compensator $\Lambda^*(t)$ defined by $\boldsymbol{F}$ must be a continuous function of t. (The compensator is already increasing and piecewise-differentiable since $\boldsymbol{F}$ is increasing and differentiable)

Condition 2. The map $\boldsymbol{F}$ is bijective (invertible) on the space of increasing sequences. In simple words, we need to ensure that for every increasing sequence $\mathcal{Z} = (z_1, ..., z_N)$ of arbitrary length N, there exists a unique increasing sequence $\mathcal{T} = (t_1, ..., t_N)$, such that $\boldsymbol{F}(\mathcal{T}) = \mathcal{Z}$.

Both of these conditions are satisfied for the models presented in Chapter 4. In the next section, we explain how these functions are implemented to make sure that they satisfy the necessary constraints.

E.1.4 Parametrizing transformations using splines

Rational quadratic splines used by Durkan et al. [53] define a flexible nonlinear function $g : (0, 1) \to (0, 1)$. When defining our TPP models in Section 4.2, we need to parametrize functions $\Lambda : [0, T] \to \mathbb{R}_+$ and $\Phi : \mathbb{R}_+ \to \mathbb{R}_+$ that operate on domains different from $(0, 1)$. Moreover, we need to ensure domain compatibility when stacking different transformations, such that the overall transformation $\boldsymbol{F}$ is bijective on the space of increasing sequences (Appendix E.1.3).

We introduce shortcuts for several helper functions that ensure the domain compatibility

1. $\boldsymbol{\psi}$ applies the function $\psi(x) = 1 - \exp(-x)$ element-wise, where $\psi : \mathbb{R}_+ \to (0, 1)$
2. $\boldsymbol{\psi}^{-1}$ applies the function $\psi^{-1}(y) = -\log(1 - y)$ element-wise, where $\psi^{-1} : (0, 1) \to \mathbb{R}_+$
3. $\boldsymbol{\sigma}$ applies the function $\sigma(x) = 1/(1 + \exp(-x))$ element-wise, where $\sigma : \mathbb{R} \to (0, 1)$
4. $\boldsymbol{\sigma}^{-1}$ applies the function $\sigma^{-1}(p) = \log p - \log(1 - p)$ element-wise, where $\sigma^{-1} : (0, 1) \to \mathbb{R}$
5. $\boldsymbol{G}$ applies a rational quadratic spline $g : (0, 1) \to (0, 1)$ element-wise.

We implement the transformation for the modulated renewal process (MRP) as

$$\boldsymbol{F} = \boldsymbol{\psi}^{-1} \circ \boldsymbol{G}_2 \circ \boldsymbol{\psi} \circ \boldsymbol{D} \circ \lambda \boldsymbol{I} \circ \boldsymbol{G}_1 \circ \frac{1}{T} \boldsymbol{I}$$

where $\boldsymbol{I}$ is the identity matrix.

Similarly, we implement the transformation for TriTPP as

$$\boldsymbol{F} = \boldsymbol{\psi}^{-1} \circ \boldsymbol{G}_3 \circ \boldsymbol{\sigma} \circ \boldsymbol{B}_L \circ \cdots \circ \boldsymbol{B}_1 \circ \boldsymbol{\sigma}^{-1} \circ \boldsymbol{G}_2 \circ \boldsymbol{\psi} \circ \boldsymbol{D} \circ \lambda \boldsymbol{I} \circ \boldsymbol{G}_1 \circ \frac{1}{T} \boldsymbol{I}$$

See the code for more details.

E.2 Datasets

For each synthetic TPP model from Omi et al. [139, Section 4.1], we sampled 1000 sequences on the interval $[0, 100]$. This includes the **Hawkes1**, **Hawkes2**, **self-correcting (SCP)**, **inhomogeneous Poisson (IPP)**, **modulated renewal (MRP)** and **renewal (RP)** processes. See Appendix D.5 for more details.

PUBG.[1] Each sequence contains timestamps of the death of players in a game of Player Unknown's Battleground (PUBG). We use the first 3001 games from the original dataset.

Reddit-Comments. Each sequence consists of the timestamps of the comments in a discussion thread posted within 24 hours of the original submission. We consider the submissions to the `/r/askscience` subreddit from 01.01.2018 until 31.12.2019. If several events happen at the *exact* same time, we only keep a single event. The posts are filtered to have a score of at least 100. We collected the data ourselves using the `pushshift` API.[2]

Reddit-Submissions. Each sequence contains the timestamps of submissions to the `/r/politics` subreddit within a single day (24 hours). We consider the period from 01.01.2017 until 31.12.2019. If several events happen at the *exact* same time, we only keep a single event. The data is again collected using the `pushshift` API.

Taxi[3] contains the records of taxi pick-ups in New York. We restrict our attention to the south of Manhattan, which corresponds to the points with latitude in the interval (40.700084, 40.707697) and longitude in (-74.019871, -73.999443).

Twitter[4] contains the timestamps of the tweets by user 25073877, recorded over several years.

Yelp 1 and 2[5] contain the user check-in times for the McCarran International Airport and for all businesses in the city of Mississauga in 2018, respectively.

Table E.1 shows the number of sequences, average sequence length and the duration of the $[0, T]$ interval for all the datasets.

E.3 Experimental setup

E.3.1 Scalability

For both the RNN-based model and TriTPP we used 20 spline knots. We ran TriTPP with blocks of size $H = 16$ and a total of $L = 4$ block-diagonal layers. This is the configuration of TriTPP with the *largest* number of parameters that we used across our experiments. For the RNN model, we used the hidden size of 32. This is the configuration of the RNN model with the *smallest* number of parameters that we used across our experiments. We did *not* use JIT compilation for either the RNN model or TriTPP, even though enabling JIT would make TriTPP even faster. When measuring the sampling time, we disabled the gradient computation with `torch.no_grad()`. To remove outliers for the RNN model, we removed 10 longest runtimes for both models.

[1] `https://kaggle.com/skihikingkevin/pubg-match-deaths`
[2] `https://pushshift.io/`
[3] `https://www.kaggle.com/c/nyc-taxi-trip-duration/data`
[4] `https://twitter.com`
[5] `https://www.yelp.com/dataset/challenge`

Table E.1: Statistics for the synthetic & real-world datasets

Dataset name	Number of sequences	Average sequence length	Interval duration
Hawkes 1	1000	95.4	100
Hawkes 2	1000	97.2	100
SCP	1000	100.2	100
IPP	1000	100.3	100
MRP	1000	98.0	100
RP	1000	109.2	100
PUBG	3001	76.5	40
Reddit-C	1356	295.7	24
Reddit-S	1094	1129.0	24
Taxi	182	98.4	24
Twitter	2019	14.9	24
Yelp 1	319	30.5	24
Yelp 2	319	55.2	24

E.3.2 Density estimation

NLL. In this experiment, we train all models by minimizing the average negative log-likelihood of the training set $\mathcal{D}_{\text{train}} = \{\mathcal{T}^{(1)}, \mathcal{T}^{(2)}, ...\}$

$$\min_{\boldsymbol{\theta}} -\frac{1}{|\mathcal{D}_{\text{train}}|}\frac{1}{N_{\text{avg}}} \sum_{\mathcal{T} \in \mathcal{D}_{\text{train}}} \log p_{\boldsymbol{\theta}}(\mathcal{T})$$

We normalize the loss by N_{avg}, the average number of events in a sequence in the training set, in order to obtain values that are at least somewhat comparable across the datasets. We perform full-batch training since the all the considered datasets easily fit into the GPU memory. For all models, we use learning rate scheduling: if the training loss does not improve for 100 iterations, the learning rate is halved. The training is stopped after 5000 epochs or if the validation loss stops improving for 300 epochs, whichever happens first. We train all models using the parameter configurations reported in Section 4.4.2 and pick the configuration with the best validation loss.

MMD. We train the models & tune the hyperparameters using the same procedure as in the NLL experiment. Then, we compare the distribution $p(\mathcal{T})$ learned by each model with the empirical distribution $p^\star(\mathcal{T})$ on the hold-out test set by estimating the maximum mean discrepancy (MMD) [76]. The MMD between distributions p and $p^\star$ is defined as

$$\text{MMD}(p, p^\star) = \mathbb{E}_{\mathcal{T},\mathcal{T}' \sim p}[k(\mathcal{T}, \mathcal{T}')] - 2\mathbb{E}_{\mathcal{T}\sim p, \mathcal{U} \sim p^\star}[k(\mathcal{T}, \mathcal{U})] + \mathbb{E}_{\mathcal{U},\mathcal{U}' \sim p^\star}[k(\mathcal{U}, \mathcal{U}')] \quad \text{(E.2)}$$

Here, $\mathcal{T} = (t_1, ..., t_N)$ and $\mathcal{U} = (u_1, ..., u_M)$ denote variable-length TPP realizations from different distributions, and $k(\cdot, \cdot)$ is a positive semi-definite kernel function that

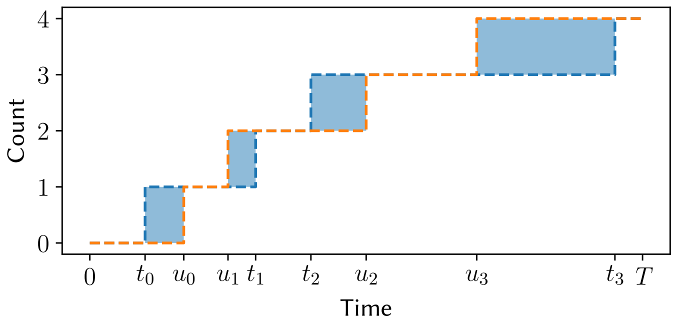


Figure E.1: The blue area represents the counting measure distance between two event sequences $\mathcal{T} = (t_1, \dots, t_N)$ and $\mathcal{U} = (u_1, ..., u_M)$ (figure adapted from [199]).

quantifies the similarity between two TPP realizations. We use the Gaussian kernel

$$k(\mathcal{T}, \mathcal{U}) = \exp\left(-\frac{d(\mathcal{T}, \mathcal{U})}{2\sigma^2}\right),$$

where $d(\mathcal{T}, \mathcal{U})$ is the counting measure distance between two TPP realizations from [199, Equation 3], defined as

$$d(\mathcal{T}, \mathcal{U}) = \sum_{i=1}^{N} |t_i - u_i| + \sum_{i=N+1}^{M} (T - u_i)$$

Here, we assume w.l.o.g. that $N \le M$. Following Section 8 of Gretton et al. [76], the parameter σ is estimated as the median of $d(\mathcal{T}, \mathcal{U})$ with $\mathcal{T}, \mathcal{U} \sim p \cup p^\star$.

E.4 Additional experiments

E.4.1 Density estimation

Effect of the block size and number on TriTPP performance. In this experiment, we show that TriTPP works well with different numbers L and sizes H of block-diagonal layers. We use the same setup as in the density estimation experiment. Table E.2 shows the test set NLL scores for different configurations. Smaller block are helpful for datasets with a clear global trend (e.g., Reddit-S, Taxi, Yelp), and larger blocks help for datasets with bursty behavior (Reddit-C, Twitter). In all cases, TriTPP is better than simpler baselines, like MRP, RP and IPP (Table 4.1).

Visualizing the effect block-diagonal matrices. A completely arbitrary compensator Λ^* leads to a completely arbitrary increasing triangular map $\boldsymbol{F}$. However, by picking a parametric class of models, such as MRP or TriTPP, we restrict the set of possible maps $\boldsymbol{F}$ that our model represent. One way to visualize the dependencies captured by the map $\boldsymbol{F}$ is by looking at its Jacobian $J_{\boldsymbol{F}}$.

Figures E.2 and E.3 show the Jacobians of the component transformations for the modulated renewal process and TriTPP. We can obtain the overall (accumulated) Jacobian of the entire transformation by multiplying the component Jacobians from right

Table E.2: Test set NLL for different configurations of TriTPP.

Configuration	Hawkes1	Hawkes2	SCP	IPP	MRP	RP	PUBG	Reddit-C	Reddit-S	Taxi	Twitter	Yelp1	Yelp2
TriTPP ($L = 2, H = 4$)	0.58	0.01	0.86	0.71	0.35	0.24	-0.95	-2.26	-4.69	-0.68	1.11	0.62	-0.1
TriTPP ($L = 4, H = 4$)	0.57	0.01	0.85	0.71	0.35	0.24	-2.04	-2.28	-4.57	-0.68	1.06	0.63	-0.1
TriTPP ($L = 2, H = 8$)	0.56	0.01	0.84	0.71	0.35	0.24	-1.93	-2.3	-4.42	-0.66	1.06	0.64	-0.09
TriTPP ($L = 4, H = 8$)	0.56	0.0	0.83	0.71	0.35	0.24	-2.41	-2.33	-4.46	-0.67	1.06	0.64	-0.09
TriTPP ($L = 2, H = 16$)	0.56	0.0	0.84	0.71	0.36	0.25	-1.78	-2.35	-4.45	-0.64	1.06	0.67	-0.06
TriTPP ($L = 4, H = 16$)	0.56	0.0	0.84	0.72	0.36	0.25	-1.83	-2.36	-4.49	-0.64	1.07	0.67	-0.06

to left. We can see that thanks to the block-diagonal layers TriTPP is able to capture more complex transformations, and thus richer densities, than MRP.

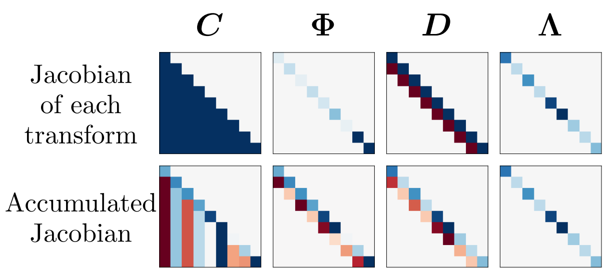


Figure E.2: Jacobians of the component transformations of the modulated renewal process. We obtain the Jacobian of the combined transformation $\boldsymbol{F} = \boldsymbol{C} \circ \boldsymbol{\Phi} \circ \boldsymbol{D} \circ \boldsymbol{\Lambda}$ by multiplying the Jacobians of each transform (right to left).

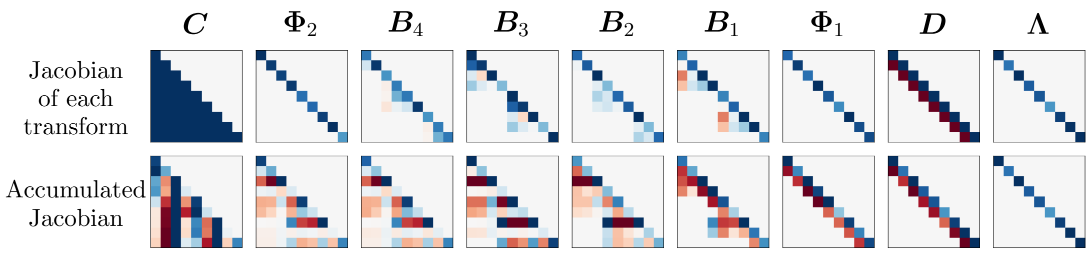
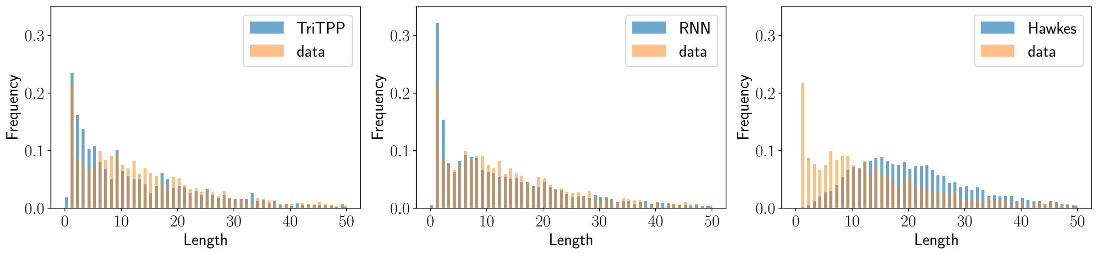


Figure E.3: Jacobians of the component transformations of TriTPP. We obtain the Jacobian of the combined transformation $\boldsymbol{F} = \boldsymbol{C} \circ \boldsymbol{\Phi}_2 \circ \boldsymbol{B}_4 \circ \boldsymbol{B}_3 \circ \boldsymbol{B}_2 \circ \boldsymbol{B}_1 \circ \boldsymbol{\Phi}_1 \circ \boldsymbol{D} \circ \boldsymbol{\Lambda}$ by multiplying the Jacobians of each transform (right to left).

Distribution of sequence lengths. In this experiment, we additionally quantify how well each model captures the true data distribution. Like before, we train all models on the training set. We then generate sequences $\mathcal{T}$ from a trained model and compare the distribution of their lengths to the distribution of the lengths of the true data using Wasserstein distance. We use the whole dataset since the test sets is too small in some



(a) TriTPP (WD = 0.17) **(b)** RNN (WD = 0.10) **(c)** Hawkes (WD = 0.50)

Figure E.4: Histograms of sequence lengths (true and generated) for Twitter. The difference between the two is quantified using Wasserstein distance (WD) — lower is better.

cases. Using Python pseudocode, this procedure can be expressed as

```
lengths_sampled = [len(t) for t in model_samples]
lengths_true = [len(t) for t in dataset]
wd = wasserstein_distance(lengths_sampled, lengths_true)
```

Figure E.4 shows the distributions for the Twitter dataset together with the respective Wasserstein distances. Note that the histograms are used only for visualization purposes, the Wasserstein distance is computed on the raw distributions. Quantitative results are reported in Table E.3. We observe the same trend as before: the RNN-based model and TriTPP consistently outperform the other methods. Recall that Hawkes process achieves a good NLL on the Twitter data (Table 4.1). However, when we sample sequences from the trained Hawkes model, the distribution of their lengths does not actually match the true data, as can be seen in Figure E.4c.

Table E.3: Wasserstein distance between the distributions of lengths of true and sampled sequences.

	Hawkes1	Hawkes2	SCP	IPP	MRP	RP	PUBG	Reddit-C	Reddit-S	Taxi	Twitter	Yelp1	Yelp2
IPP	0.11	0.11	0.03	**0.00**	0.03	0.07	**0.01**	0.76	0.27	0.10	0.52	0.07	0.12
RP	0.07	0.09	0.19	0.14	0.38	0.02	0.67	0.67	0.28	0.85	0.28	0.31	0.20
MRP	0.08	0.07	0.02	**0.00**	**0.01**	0.01	0.05	0.66	0.27	0.09	0.28	0.06	0.11
Hawkes	0.01	0.05	0.03	0.15	0.15	0.03	0.12	**0.25**	0.65	0.09	0.50	0.10	0.15
RNN	**0.00**	**0.01**	**0.00**	0.04	0.03	**0.00**	0.08	0.40	**0.07**	**0.08**	**0.10**	**0.05**	0.12
TriTPP	0.05	0.03	**0.00**	0.01	**0.01**	**0.00**	0.05	0.53	0.24	**0.08**	0.17	**0.05**	**0.09**

F Supplementary materials for Chapter 5

F.1 Model definition

F.1.1 Markov jump process (MJP)

We represent the trajectory of an MJP as a tuple $(\mathcal{T}, \mathcal{S})$, where $\mathcal{T} = (t_1, ..., t_N)$ are the (strictly increasing) jump times and $\mathcal{S} = (s_1, ..., s_{N+1})$ is the sequence of visited states. For convenience, we additionally set $t_0 = 0$ and $t_{N+1} = T$.

The distribution over the trajectories $(\mathcal{T}, \mathcal{S})$ is defined by a $K \times K$ nonnegative generator matrix $\boldsymbol{A}$ and an initial state distribution $\boldsymbol{\pi}$. Each entry A_{kl} denotes the rate of transition from state k to state l of the MJP. Note that we use the formulation that permits self-jumps [151] (i.e., it may happen that $s_i = s_{i+1}$). We denote the total transition rate of state s_i as $A_{s_i} = \sum_{k=1}^{K} A_{s_i k}$. We can simulate an MJP trajectory using the following procedure

$$\begin{aligned} &s_1 \sim \text{Categorical}(\boldsymbol{\pi}) \\ &t_i - t_{i-1} =: \tau_i \sim \text{Exponential}\left(A_{s_i}\right) \\ &s_{i+1} \sim \text{Categorical}\left(\boldsymbol{A}_{s_i:}/A_{s_i}\right) \end{aligned} \tag{F.1}$$

Here $\boldsymbol{A}_{s_i:}/A_{s_i}$ is the s_i'th row of matrix $\boldsymbol{A}$ that is normalized to sum up to 1.

According to the above generative process, the likelihood of a trajectory $(\mathcal{T}, \mathcal{S})$ for an MJP with parameters $(\boldsymbol{\pi}, \boldsymbol{A})$ can be computed as

$$p(\mathcal{T}, \mathcal{S} | \boldsymbol{\pi}, \boldsymbol{A}) = \pi_{s_1} \times \left(\prod_{i=1}^{N} A_{s_{i-1} s_i} \right) \times \exp \left(- \sum_{i=1}^{N+1} (t_i - t_{i-1}) \sum_{l=1}^{K} A_{s_i l} \right)$$

We reformulate this expression using indicator functions $\mathbb{1}(\cdot)$, which will be allow us to derive a differentiable relaxation later

$$\begin{aligned} &= \left(\prod_{k=1}^{K} \pi_k^{\mathbb{1}(s_1 = k)} \right) \times \left(\prod_{i=1}^{N} \prod_{k=1}^{K} \prod_{l=1}^{K} A_{kl}^{\mathbb{1}(s_i = k, s_{i+1} = l)} \right) \\ &\quad \times \exp \left(- \sum_{i=1}^{N+1} (t_i - t_{i-1}) \sum_{k=1}^{K} \mathbb{1}(s_i = k) \left(\sum_{l=1}^{K} A_{kl} \right) \right). \end{aligned}$$

By applying the logarithm to the above equation, we obtain

$$\begin{aligned}\log p(\mathcal{T},\mathcal{S}|\boldsymbol{\pi},\boldsymbol{A}) = &\left(\sum_{k=1}^{K}\mathbb{1}(s_1=k)\log\pi_k\right)\\ &+\left(\sum_{i=1}^{N}\sum_{k=1}^{K}\sum_{l=1}^{K}\mathbb{1}(s_i=k,s_{i+1}=l)\log A_{kl}\right)\\ &-\left(\sum_{i=1}^{N+1}(t_i-t_{i-1})\sum_{k=1}^{K}\mathbb{1}(s_i=k)\left(\sum_{l=1}^{K}A_{kl}\right)\right).\end{aligned}\tag{F.2}$$

F.1.2 Markov modulated Poisson process (MMPP)

MMPP consists of a latent MJP $p(\mathcal{T},\mathcal{S})$ and observed events $\mathcal{X}=(x_1,...,x_M)$. The distribution of $\mathcal{X}$ is governed by unobserved state of the MJP $s(t)$ and the emission rates for each state $\boldsymbol{\lambda}\in\mathbb{R}_+^K$. More specifically, the observations $\mathcal{X}$ are sampled from an inhomogeneous Poisson process with piecewise-constant intensity that depends on the current state: $\lambda(t)=\lambda_{s(t)}$.

Likelihood of the observations $\mathcal{X}$ given $(\mathcal{T},\mathcal{S})$ and $\boldsymbol{\lambda}$ can be computed as

$$p(\mathcal{X}|\mathcal{T},\mathcal{S},\boldsymbol{\lambda}) = \left(\prod_{i=1}^{N+1}\lambda_{s_i}^{M_{[t_{i-1},t_i)}}\right)\times\exp\left(-\sum_{i=1}^{N+1}(t_i-t_{i-1})\lambda_{s_i}\right)$$

where $M_{[t_{i-1},t_i)}$ is the number of events x_j in the interval $[t_{i-1},t_i)$. Again, using indicator functions, we rewrite it as

$$\begin{aligned}&=\left(\prod_{i=1}^{N+1}\prod_{j=1}^{M}\left(\prod_{k=1}^{K}\lambda_k^{\mathbb{1}(s_i=k)}\right)^{\mathbb{1}(x_j\in[t_{i-1},t_i))}\right)\\ &\quad\times\exp\left(-\sum_{i=1}^{N+1}(t_i-t_{i-1})\sum_{k=1}^{K}\mathbb{1}(s_i=k)\lambda_k\right)\end{aligned}$$

By applying the logarithm, we obtain

$$\begin{aligned}\log p(\mathcal{X}|\mathcal{T},\mathcal{S},\boldsymbol{\lambda}) = &\left(\sum_{i=1}^{N+1}\sum_{j=1}^{M}\mathbb{1}(x_j\in[t_{i-1},t_i))\sum_{k=1}^{K}\mathbb{1}(s_i=k)\log\lambda_k\right)\\ &-\left(\sum_{i=1}^{N+1}(t_i-t_{i-1})\sum_{k=1}^{K}\mathbb{1}(s_i=k)\lambda_k\right)\end{aligned}\tag{F.3}$$

To summarize, an MMPP is governed by parameters $\boldsymbol{\psi}=\{\boldsymbol{\pi},\boldsymbol{A},\boldsymbol{\lambda}\}$. The initial state probabilities $\boldsymbol{\pi}$ and the generator $\boldsymbol{A}$ define a prior over the (unobserved) trajectories $p(\mathcal{T},\mathcal{S}|\boldsymbol{\pi},\boldsymbol{A})$, and the emission rates $\boldsymbol{\lambda}$ governs the distribution over the observed events $p(\mathcal{X}|\mathcal{T},\mathcal{S},\boldsymbol{\lambda})$.

F.1.3 Derivation of the ELBO

We consider the following problem: Given the observed events $\mathcal{X}$ and the number of latent states K, we want to infer the unobserved trajectories $(\mathcal{T}, \mathcal{S})$. According to the Bayes theorem, the posterior over trajectories of an MMPP is computed as

$$\begin{aligned} p(\mathcal{T}, \mathcal{S} | \mathcal{X}, \boldsymbol{\pi}, \boldsymbol{A}, \boldsymbol{\lambda}) &\propto p(\mathcal{T}, \mathcal{S}, \mathcal{X} | \boldsymbol{\pi}, \boldsymbol{A}, \boldsymbol{\lambda}) \\ &= p(\mathcal{T}, \mathcal{S} | \boldsymbol{\pi}, \boldsymbol{A}) p(\mathcal{X} | \mathcal{T}, \mathcal{S}, \boldsymbol{\lambda}). \end{aligned}$$

Unfortunately, this quantity is intractable, so we approximate it with a variational distribution $q(\mathcal{T}, \mathcal{S})$. We find the "optimal" approximate posterior by maximizing the evidence lower bound (ELBO) [207]

$$\text{ELBO}(q, \boldsymbol{\psi}) = \mathbb{E}_{q(\mathcal{T}, \mathcal{S})} \Bigg[\underbrace{\log p(\mathcal{T}, \mathcal{S} | \boldsymbol{\pi}, \boldsymbol{A})}_{\text{trajectory log-likelihood}} + \underbrace{\log p(\mathcal{X} | \mathcal{T}, \mathcal{S}, \boldsymbol{\lambda})}_{\text{observations log-likelihood}} \underbrace{- \log q(\mathcal{T}, \mathcal{S})}_{\text{entropy}} \Bigg]$$

Since we are interested only in q, we assume that the model parameters $\boldsymbol{\psi} = \{\boldsymbol{\pi}, \boldsymbol{A}, \boldsymbol{\lambda}\}$ are known. In the next section, we will discuss how the parameters $\boldsymbol{\psi}$ can be learned as well.

As described in Section 5.3 we model the approximate posterior as $q(\mathcal{T}, \mathcal{S}) = q(\mathcal{T}) q(\mathcal{S} | \mathcal{T})$. The distribution $q(\mathcal{T})$ over the jump times is defined using TriTPP, and $q(\mathcal{S} | \mathcal{T})$ is evaluated exactly for each Monte Carlo sample $\mathcal{T}$. We rewrite the ELBO as

$$\text{ELBO}(q, \boldsymbol{\psi}) = \mathbb{E}_{q(\mathcal{T})} \left[\mathbb{E}_{q(\mathcal{S}|\mathcal{T})} \left[\log p(\mathcal{T}, \mathcal{S} | \boldsymbol{\pi}, \boldsymbol{A}) + \log p(\mathcal{X} | \mathcal{T}, \mathcal{S}, \boldsymbol{\lambda}) - \log q(\mathcal{S} | \mathcal{T}) \right] - \log q(\mathcal{T}) \right]$$

We already derived the expressions for $\log p(\mathcal{T}, \mathcal{S} | \boldsymbol{\pi}, \boldsymbol{A})$ (Equation F.2) and $\log p(\mathcal{X} | \mathcal{T}, \mathcal{S}, \boldsymbol{\lambda})$ (Equation F.3). The expression for $\log q(\mathcal{S} | \mathcal{T})$ can be obtained similarly as

$$\log q(\mathcal{S} | \mathcal{T}) = \sum_{i=1}^{N+1} \sum_{k=1}^{K} \mathbb{1}(s_i = k) \log q(s_i = k | \mathcal{T}). \tag{F.4}$$

Finally, we compute the log-density $\log q(\mathcal{T})$ of a single sample $\mathcal{T} = (t_1, ..., t_N)$ using the change of variables formula, as described in Appendix E.1. We denote $\mathcal{Z} = (z_1, ..., z_N, z_{N+1}) = \boldsymbol{F}(t_1, ..., t_N, T)$, and z_{-1} denotes the last entry of $\mathcal{Z}$.

$$\log q(\mathcal{T}) = \sum_{i=1}^{N} \log \left| \frac{\partial z_i}{\partial t_i} \right| - z_{-1}. \tag{F.5}$$

ELBO (non-differentiable version). Putting everything together, we get

$$\begin{aligned}
\mathrm{ELBO}(q, \boldsymbol{\psi}) = \mathbb{E}_{q(\mathcal{T})}\Bigg[\mathbb{E}_{q(\mathcal{S}|\mathcal{T})}\Bigg[&\sum_{k=1}^{K} \mathbb{1}(s_1 = k) \log \pi_k \\
&- \sum_{i=1}^{N+1} (t_i - t_{i-1}) \sum_{k=1}^{K} \mathbb{1}(s_i = k) \sum_{l=1}^{K} A_{kl} \\
&+ \sum_{i=1}^{N} \sum_{k=1}^{K} \sum_{l=1}^{K} \mathbb{1}(s_i = k, s_{i+1} = l) \log A_{kl} \\
&+ \sum_{i=1}^{N+1} \sum_{j=1}^{M} \mathbb{1}(x_j \in [t_i, t_{i+1})) \sum_{k=1}^{K} \mathbb{1}(s_i = k) \log \lambda_k \\
&- \sum_{i=1}^{N+1} (t_i - t_{i-1}) \sum_{k=1}^{K} \mathbb{1}(s_i = k) \lambda_k \\
&- \sum_{i=1}^{N+1} \sum_{k=1}^{K} \mathbb{1}(s_i = k) \log q(s_i = k|\mathcal{T})\Bigg] \\
&- \sum_{i=1}^{N} \log \left|\frac{\partial z_i}{\partial t_i}\right| + z_{-1}\Bigg]
\end{aligned} \tag{F.6}$$

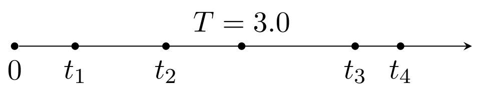
Note that N is the length of the sample $\mathcal{T}$ generated from $q(\mathcal{T})$, so N will take different values for different samples. We can evaluate the inner expectation w.r.t. $q(\mathcal{S}|\mathcal{T})$ by using the following two facts

$$\begin{aligned}
\mathbb{E}_{q(\mathcal{S}|\mathcal{T})}\left[\mathbb{1}(s_i = k)\right] &= q(s_i = k|\mathcal{T}) \\
\mathbb{E}_{q(\mathcal{S}|\mathcal{T})}\left[\mathbb{1}(s_i = k, s_{i+1} = l)\right] &= q(s_i = k, s_{i+1} = l|\mathcal{T})
\end{aligned}$$

Recall that we set $q(\mathcal{S}|\mathcal{T})$ to the true posterior $p(\mathcal{S}|\mathcal{T}, \mathcal{X}, \boldsymbol{\psi})$ over states given the jumps (Equation 5.15). This allows us to exactly compute both the posterior marginals $q(s_i = k|\mathcal{T})$ and the posterior transition probabilities $q(s_i = k, s_{i+1} = l|\mathcal{T})$ using the forward-backward algorithm [188, Equations 7, 8]. Therefore, the inner expectation w.r.t. $q(\mathcal{S}|\mathcal{T})$ in Equation F.6 can be computed analytically.

ELBO (differentiable relaxation). The ELBO, as defined above in Equation F.6, is discontinuous w.r.t. the parameters of the density $q(\mathcal{T})$ for the reasons described in Section 5.2.2. The expression inside the expectation depends only on the events t_i that happen before T. Infinitesimal change in the parameters of $q(\mathcal{T})$ may "push" the point t_i outside $[0, T]$, thus changing the function value by a fixed amount and resulting in a discontinuity.

We fix this problem using the approach described in Appendix E.1 and Section 5.2.2. We obtain an "extended" sample $\tilde{\boldsymbol{t}}$ by first simulating a sequence $\tilde{\boldsymbol{z}} = (\tilde{z}_1, ..., \tilde{z}_{N'})$ from a HPP with unit rate and computing $\tilde{\boldsymbol{t}} = \boldsymbol{F}^{-1}(\tilde{\boldsymbol{z}})$ (more on this in Appendix E.1). We get a "clipped" / "padded" sample $\boldsymbol{t} = (t_1, ..., t_{N'})$ as $t_i = \min\{\tilde{t}_i, T\}$ (Figure F.1).



Extended sample	$\tilde{\boldsymbol{t}}$	(0.8, 2.0, 4.5, 5.1)
Clipped sample	$\boldsymbol{t}$	(0.8, 2.0, 3.0, 3.0)
Hard indicator	$\mathbb{1}(t_i < T)$	(1.00, 1.00, 0.00, 0.00)
Relaxed indicator	$\sigma_\zeta(T - t_i)$	(1.00, 0.97, 0.01, 0.00)

Figure F.1: Examples of values involved in the ELBO computation.

Finally, we compute $\boldsymbol{z} = (z_1, ..., z_{N'}) = \boldsymbol{F}(\boldsymbol{t})$ (this is necessary for computing the correct cumulative intensity $\Lambda^*(T)$ after clipping). We can now express the ELBO in terms of the "extended" samples $\tilde{\boldsymbol{t}}$ and "clipped" samples $\boldsymbol{t}$:

$$
\begin{aligned}
\operatorname{ELBO}(q, \boldsymbol{\psi}) = \mathbb{E}_{q(\mathcal{T})}\Bigg[\mathbb{E}_{q(\mathcal{S}|\mathcal{T})}\Bigg[&\sum_{k=1}^{K} \mathbb{1}(s_1 = k) \log \pi_k \\
&- \sum_{i=1}^{N'} (t_i - t_{i-1}) \sum_{k=1}^{K} \mathbb{1}(s_i = k) \sum_{l=1}^{K} A_{kl} \\
&+ \sum_{i=1}^{N'} \mathbb{1}(\tilde{t}_i < T) \sum_{k=1}^{K} \sum_{l=1}^{K} \mathbb{1}(s_i = k, s_{i+1} = l) \log A_{kl} \\
&+ \sum_{i=1}^{N'} \sum_{j=1}^{M} \mathbb{1}(x_j \in [t_i, t_{i+1})) \sum_{k=1}^{K} \mathbb{1}(s_i = k) \log \lambda_k \\
&- \sum_{i=1}^{N'} (t_i - t_{i-1}) \sum_{k=1}^{K} \mathbb{1}(s_i = k) \lambda_k \\
&- \sum_{i=1}^{N'} \mathbb{1}(\tilde{t}_{i-1} < T) \sum_{k=1}^{K} \mathbb{1}(s_i = k) \log q(s_i = k|\mathcal{T}) \Bigg] \\
&- \sum_{i=1}^{N'} \mathbb{1}(\tilde{t}_i < T) \log \left|\frac{\partial z_i}{\partial t_i}\right| + z_{-1} \Bigg]
\end{aligned} \tag{F.7}
$$

Changes from Equation F.6 are highlighted in blue. Even though the formula looks different, the result of evaluating Equation F.7 will be *exactly* the same as for Equation F.6. By using different notation we only made the process of "discarding" the events $t_i > T$ explicit. The new formulation allows us to obtain a differentiable relaxation. For this, we replace the indicator functions $\mathbb{1}(t_i < T)$ with sigmoids $\sigma_\zeta(T - t_i)$. The indicator function $\mathbb{1}(x_j \in [t_i, t_{i+1}))$ can also be relaxed as

$$
\begin{aligned}
\mathbb{1}(x_j \in [t_i, t_{i+1})) &= \mathbb{1}(t_{i+1} > x_j) - \mathbb{1}(t_i \geq x_j) \\
&\approx \sigma_\zeta(t_{i+1} - x_j) - \sigma_\zeta(t_i - x_j)
\end{aligned} \tag{F.8}
$$

By combining all these facts, we obtain a differentiable relaxation of the ELBO. Our method leads to an efficient implementation that uses batches of samples. We sample a batch of jump times $\{\mathcal{T}^{(1)}, \mathcal{T}^{(2)}, ...\}$ from $q(\mathcal{T})$, evaluate the posterior $q(\mathcal{S}|\mathcal{T})$ using with forward-backward for all of them in parallel, and evaluate the relaxed ELBO (Equation F.7).

F.1.4 Parameter estimation

In Section 5.3, we perform approximate posterior inference over the trajectories $(\mathcal{T}, \mathcal{S})$ by maximizing the ELBO w.r.t. $q(\mathcal{T}, \mathcal{S})$

$$\max_{q(\mathcal{T},\mathcal{S})} \mathbb{E}_q[\log p(\mathcal{T}, \mathcal{S}|\boldsymbol{\psi}) + \log p(\mathcal{X}|\mathcal{T}, \mathcal{S}, \boldsymbol{\psi}) - \log q(\mathcal{T}, \mathcal{S})] \tag{F.9}$$

Since $\text{ELBO}(q, \boldsymbol{\psi})$ provides a lower bound on the marginal log-likelihood $\log p(\mathcal{X}|\boldsymbol{\psi})$, we can also simultaneously learn the model parameters $\boldsymbol{\psi} = \{\boldsymbol{\pi}, \boldsymbol{A}, \boldsymbol{\lambda}\}$ by solving the following optimization problem (subject to appropriate constraints on $\boldsymbol{\psi}$)

$$\max_{\boldsymbol{\psi}} \max_{q(\mathcal{T},\mathcal{S})} \mathbb{E}_q[\log p(\mathcal{T}, \mathcal{S}|\boldsymbol{\psi}) + \log p(\mathcal{X}|\mathcal{T}, \mathcal{S}, \boldsymbol{\psi}) - \log q(\mathcal{T}, \mathcal{S})] \tag{F.10}$$

Finally, we can perform fully Bayesian treatment and approximate the posterior distribution over the parameters as well as the trajectories. For this, we can place a prior $p(\boldsymbol{\psi})$ and approximate $p(\boldsymbol{\psi}, \mathcal{T}, \mathcal{S}|\mathcal{X})$ with $q(\boldsymbol{\psi}, \mathcal{T}, \mathcal{S}) = q(\boldsymbol{\psi})q(\mathcal{T})q(\mathcal{S}|\mathcal{T}, \boldsymbol{\psi})$. This corresponds to the following optimization problem

$$\max_{q(\boldsymbol{\psi},\mathcal{T},\mathcal{S})} \mathbb{E}_q[\log p(\mathcal{T}, \mathcal{S}|\boldsymbol{\psi}) + \log p(\mathcal{X}|\mathcal{T}, \mathcal{S}, \boldsymbol{\psi}) - \log q(\mathcal{T}, \mathcal{S})] - \mathbb{KL}(q(\boldsymbol{\psi})\|p(\boldsymbol{\psi})) \tag{F.11}$$

where $\mathbb{KL}$ denotes KL-divergence. By applying our relaxation from Section 5.2.2, it is possible to solve all of the above optimization problems (Equations F.9, F.10, F.11) using gradient ascent.

F.2 Experimental setup

We simulate an MMPP with $K = 3$ states and the following parameters

$$\boldsymbol{A} = \begin{pmatrix} 0.1 & 0.1 & 0.1 \\ 0.1 & 0.1 & 0.1 \\ 0.1 & 0.1 & 0.1 \end{pmatrix} \qquad \boldsymbol{\pi} = \begin{pmatrix} 0.52 \\ 0.22 \\ 0.26 \end{pmatrix} \qquad \boldsymbol{\lambda} = \begin{pmatrix} 1 \\ 5 \\ 20 \end{pmatrix}$$

We use the following configuration for TriTPP in this experiment: $L = 2$ blocks of size $H = 4$, learning rate 0.01, no weight decay. We estimate the ELBO using 512 Monte Carlo samples from $q(\mathcal{T})$ and use the temperature $\zeta = 0.1$ for the relaxation.

We implemented the MCMC sampler by Rao & Teh [149] in Pytorch. We discard the first 100 samples (burn-in stage), and use 1000 samples to compute the marginal distribution of the posterior.

G Supplementary materials for Chapter 6

G.1 Difference between GoF testing and OoD detection

The connection between OoD detection and GoF testing was first pointed out by Nalisnick et al. [128]. They proposed to perform a GoF test for a deep generative model to detect OoD instances. However, as we explained in Section 6.1, these two problems are in fact *not* equivalent. We now demonstrate how this insight allows us to explain and improve upon some results obtained by Nalisnick et al. [128].

First, we consider the **Gaussian annulus test** for normalizing flow models that was also used by Choi et al. [32]. A normalizing flow model $\mathbb{P}_{\text{model}}$ defines the distribution of a D-dimensional random vector X by specifying a diffeomorphism $f\colon \mathbb{R}^D \to \mathbb{R}^D$, such that $Z = f(X)$ is distributed according to $\mathcal{N}(\mathbf{0}_D, \boldsymbol{I}_D)$, the standard normal distribution. In other words, $f(X)|X \sim \mathbb{P}_{\text{model}}$ follows the standard normal distribution, so any test for the normal distribution can be used to test the GoF of a normalizing flow model. Based on this, Nalisnick et al. [128] define the following test statistic

$$\phi(X) = \Big| \|f(X)\|_2 - \mathbb{E}_{X\sim\mathbb{P}_{\text{model}}}[\|f(X)\|_2] \Big| = \Big| \|f(X)\|_2 - \sqrt{D} \Big|. \tag{G.1}$$

The idea here is to replace a two-sided test on the statistic $\|f(X)\|_2$ with a one-sided test on the statistic $\phi(X)$ defined above. Since $f(X)|X \sim \mathbb{P}_{\text{model}}$ follows the standard normal distribution, the statistic $\phi(X)|H_0$ will concentrate near 0 [19, Theorem 2.9]. Therefore, checking if $\phi(X)$ is below a certain threshold ϵ is equivalent to performing the GoF null hypothesis test (Equation 6.2).

However, the above approach will not work for an OoD detection hypothesis test (Equation 6.1). If we learn a model $\mathbb{P}_{\text{model}}$ on training instances $\mathcal{D}_{\text{train}}$ that were generated by some distribution $\mathbb{P}_{\text{data}}$, we will in general have $\mathbb{P}_{\text{model}} \neq \mathbb{P}_{\text{data}}$. This implies that $f(X)|X \sim \mathbb{P}_{\text{data}}$ will not follow the standard normal distribution. Therefore, $\mathbb{E}_{X\sim\mathbb{P}_{\text{data}}}[\|f(X)\|_2] \neq \sqrt{D}$ and the distribution of $\|f(X)\|_2$ might not even be symmetric around its mean. This means we cannot replace a two-sided test on $\|f(X)\|_2$ with a one-sided test on $\phi(X)$ when doing OoD detection. A better idea is to directly compute the two-sided p-value for the OoD detection test using the statistic $\|f(X)\|_2$, following our approach in Section 6.1.

Similarly, for the (single-instance) **typicality test**, the test statistic is defined as

$$\varphi(X) = \Big| \log q(X) - \mathbb{E}_{X\sim\mathbb{P}_{\text{model}}}[\log q(X)] \Big|, \tag{G.2}$$

where $\log q(X)$ is the log-likelihood of a generative model trained on $\mathcal{D}_{\text{train}}$. This leads to the same problems when trying to apply this statistic for OoD detection as we encountered with the Gaussian annulus test above—the expected value $\mathbb{E}_{X\sim\mathbb{P}_{\text{model}}}[\log q(X)]$ is only suitable for a GoF test. However, in this case Nalisnick et al. [128] report that they found $\mathbb{E}_{X\sim\mathbb{P}_{\text{data}}}[\log q(X)]$ to work better in practice. By drawing a clear distinction between the OoD detection test and the GoF test we can explain this empirical result. An even better idea is to use the two-sided p-value (Equation 6.3) instead of Equation G.2, since the distribution of the statistic $\log q(X)|X \sim \mathbb{P}_{\text{data}}$ is not guaranteed to be symmetric.

G.2 Other statistics based on squared spacings

The following discussion is based on Moran [123] and D'Agostino [40].

Sum-of-squared spacings (3S) statistic for $\text{Uniform}([0,1])$**.** Suppose that samples $\{u_1, \dots, u_N\}$ are drawn i.i.d. from the $\text{Uniform}([0,1])$ distribution. Additionally, assume w.l.o.g. that the u_i's are sorted in an increasing order, i.e., $u_1 \le \dots \le u_N$. The 3S statistic for $\text{Uniform}([0,1])$ is defined as

$$\psi_N^{\text{Unif}([0,1])} = N \sum_{i=1}^{N+1} (u_i - u_{i-1})^2, \tag{G.3}$$

where $u_0 = 0$ and $u_{N+1} = 1$. The factor N ensures that $\psi_N^{\text{Unif}([0,1])}$ approaches the standard normal distribution as $N \to \infty$. However, the convergence of $\psi_N^{\text{Unif}([0,1])}$ to its limiting distribution is rather slow.

3S statistic for $\text{Uniform}([0,V])$**.** The statistic above can be generalized to the uniform distribution on an arbitrary interval $[0, V]$. Suppose $\{v_1, \dots, v_N\}$ are drawn i.i.d. from the $\text{Uniform}([0,V])$ distribution, and again are sorted in an increasing order. The 3S statistic for $\text{Uniform}([0,V])$ is defined by simply dividing the v_i's by the interval length V.

$$\begin{aligned}\psi_N^{\text{Unif}([0,V])} &= N \sum_{i=1}^{N+1} \left(\frac{v_i}{V} - \frac{v_{i-1}}{V}\right)^2 \\ &= \frac{N}{V^2} \sum_{i=1}^{N+1} (v_i - v_{i-1})^2,\end{aligned} \tag{G.4}$$

where $v_0 = 0$ and $v_{N+1} = V$.

3S statistic for the SPP on $[0, V]$**.** Remember that the N factor makes the distribution of $\psi_N^{\text{Unif}([0,V])}$ (asymptotically) invariant for different values of N. This means that such statistic would not be able to detect anomalies in terms of the event count. To remove this undesirable property, we define the *3S statistic for the standard Poisson* *process* by replacing N with its expectation $\mathbb{E}[N|V] = V$.

$$\begin{aligned}\psi^{\text{SPP}([0,V])} &= \frac{\mathbb{E}[N|V]}{V^2} \sum_{i=1}^{N+1} (v_i - v_{i-1})^2 \\ &= \frac{1}{V} \sum_{i=1}^{N+1} (v_i - v_{i-1})^2\end{aligned} \tag{G.5}$$

This is the definition that we introduced in Equation 6.7. As a side note, replacing N with $\mathbb{E}[N|V]$ is one of the possible choices that ensures that (1) the statistic is sensitive to changes in the event count and (2) the expected value $\mathbb{E}[\psi^{\text{SPP}([0,V])}|V]$ does not change for different values of V, and hence is comparable across different transformed sequences.

As we show in Sections 6.3 and 6.5, the above definition of the TriTPP statistic for the SPP allows us to detect a broad class of anomalies (i.e., deviations from the SPP) that differ both in the distribution of the event count N as well as the arrival times v_i.

G.3 Proof of Proposition 1

To compute the moments of the 3S statistic for the standard Poisson process (Equation 6.7) we need to marginalize out the event count N, which is equivalent to applying the law of iterated expectation

$$\begin{aligned}\mathbb{E}[f(\psi)|T] &= \sum_{n=0}^{\infty} \mathbb{E}[f(\psi)|N = n, V] \Pr(N = n|V) \\ &= \sum_{n=0}^{\infty} \mathbb{E}[f(\psi)|N = n, V] \frac{V^n e^{-V}}{n!}\end{aligned} \tag{G.6}$$

where we used the fact that $N|V \sim \text{Poisson}(V)$.

We obtain the expectations of ψ and ψ^2 conditioned on N and V using the result by Moran [123] on the moments of $\psi_N^{\text{Unif}([0,1])}$ (Equation G.3), the TriTPP statistic for the Uniform($[0, 1]$) distribution.

$$\begin{aligned}\mathbb{E}[\psi|N = n, V] &= \frac{2V}{(n + 2)} \\ \mathbb{E}[\psi^2|N = n, V] &= \frac{4V^2(n + 6)}{(n + 2)(n + 3)(n + 4)}\end{aligned} \tag{G.7}$$

These can also be easily derived from the moments of the Dirichlet distribution, by using the fact that the scaled inter-event times $({w_1}/{V}, \dots, {w_{N+1}}/{V})$ are distributed uniformly on the standard N-simplex (i.e., according to Dirichlet distribution with parameter $\boldsymbol{\alpha} = \mathbf{1}_{N+1}$).

By plugging in Equation G.7 into Equation G.6, we obtain

$$\begin{aligned}\mathbb{E}[\psi|V] &= 2Ve^{-V}\sum_{n=0}^{\infty}\frac{V^n}{n!(n+2)} \\ &= 2Ve^{-V}\frac{1}{V^2}\left(e^V(V-1)+1\right) \\ &= \frac{2}{V}(V+e^{-V}-1).\end{aligned} \tag{G.8}$$

Similarly, we compute the non-centered second moment as

$$\begin{aligned}\mathbb{E}[\psi^2|V] &= 4V^2e^{-V}\sum_{n=0}^{\infty}\frac{V^n(n+6)}{n!(n+2)(n+3)(n+4)} \\ &= 4V^2e^{-V}\frac{1}{V^4}\left(e^V(V^2-6)+2(V^2+3V+3)\right) \\ &= \frac{4}{V^2}\left(V^2-6+2e^{-V}(V^2+3V+3)\right).\end{aligned} \tag{G.9}$$

Finally, we obtain the variance as

$$\begin{aligned}\mathrm{Var}[\psi|V] &= \mathbb{E}[\psi^2|V]-\mathbb{E}[\psi|V]^2 \\ &= \frac{4}{V^2}\left(2V-7+e^{-V}(2V^2+4V+8-e^{-V})\right).\end{aligned}$$

Higher-order moments of $\psi|V$ can be computed similarly using Equation G.6.

G.4 Implementation details

The following code describes the procedure for computing the p-values for both hypothesis tests discussed in Section 6.1—namely, the GoF test (Equation 6.2) and the OoD detection test (Equation 6.1). The code below is for demonstration purposes only, the actual implementation used in our experiments is better optimized.

```python
def compute_p_value(x_test, samples, score_fn):
    scores_id = [score_fn(x) for x in samples]
    score_x = score_fn(x_test)
    num_train = len(samples)
    num_above = 0
    for s in scores_id:
        if s > score_x:
            num_above += 1
    num_below = num_train - num_above
    return min(
        (num_below + 1) / (num_train + 1),
        (num_above + 1) / (num_train + 1)
    )
```

The `+1` correction in the numerator and denominator for the p-value computation is done as described by North et al. [131]. If we define `samples` as the set of in-distribution sequences $\mathcal{D}_{\text{train}}$ that were generated from $\mathbb{P}_{\text{data}}$, we recover the OoD detection test (Equation 6.1 and Section 6.3.2). If we define `samples` as the set of sequences $\mathcal{D}_{\text{model}}$ that were generated from $\mathbb{P}_{\text{model}}$, we recover the GoF test (Equation 6.2 and Section 6.3.1).

In the snippet above, `score_fn` corresponds to a test statistic $s\colon \mathcal{X} \to \mathbb{R}$. In our experiments, we consider the following choices for s:

1. KS arrival (Equation 6.5).
2. KS inter-event (Equation 6.6).
3. Chi-squared: we partition the interval $[0, V]$ into $B = 10$ disjoint buckets of equal length, and compare the observed event count N_b in each bucket with the expected amount $L = V/B$

$$\chi^2(Z) = \sum_{b=1}^{B} \frac{(N_b - L)^2}{L}. \tag{G.10}$$

4. Sum-of-squared spacings (Equation 6.7).
5. Log-likelihood

$$\log q(X) = \sum_{i=1}^{N} \log \frac{\partial \Lambda^*(t_i)}{\partial t_i} - \Lambda^*(T). \tag{G.11}$$

All these statistics are computed based on some TPP model with compensator Λ^*. For statistic 1–4, we compute $s(X)$ by first obtaining the transformed sequence $Z = (\Lambda^*(t_1), \dots, \Lambda^*(T))$ and then evaluating the respective SPP statistic on Z. The log-likelihood is directly evaluated based on the model's conditional intensity.

Marked sequences. In a marked sequence $X = \{(t_1, m_1), \dots, (t_N, m_N)\}$ each event is represented by a categorical mark $m_i \in \{1, \dots, C\}$ in addition to the arrival time t_i. A marked TPP model is specified by C compensators $\{\Lambda_1^*, \dots, \Lambda_C^*\}$.

We obtain the transformed sequence Z necessary for statistics 1–4 as follows. Let $\left(t_1^{(c)}, \dots, t_{N_c}^{(c)}\right)$ denote the events of mark c in a given sequence X. For each event type $c \in \{1, \dots, C\}$, we obtain a transformed sequence $Z^{(c)} = \left(\Lambda_c^*(t_1^{(c)}), \dots, \Lambda_c^*(t_{N_c}^{(c)}), \Lambda_c^*(T)\right)$. Then we concatenate the transformed sequences for each mark, thus obtaining a single SPP realization on the interval $[0, \sum_{c=1}^{C} \Lambda_c^*(T)]$. For example, suppose the transformed sequence for the first mark is $Z^{(1)} = (1.0, 2.5, 4.0)$ and for the second mark $Z^{(2)} = (0.5, 3.0)$. Then the concatenated sequence will be $Z = (0.0, 1.0, 2.5, 4.0{+}0.5, 4.0{+}3.0) = (0.0, 1.0, 2.5, 4.5, 7.0)$. Our approach based on concatenating the $Z^(c)$'s is simpler than other methods for combining multiple sequences by Gerhard et al. [66] & Tao et al. [179], and we found ours to work well in practice.

The log-likelihood for a marked sequence is computed as

$$\log q(X) = \sum_{c=1}^{C} \sum_{i=1}^{N} \mathbb{1}(m_i = c) \log \frac{d\Lambda_c^*(t_i)}{dt_i} - \sum_{c=1}^{C} \Lambda_c^*(T). \tag{G.12}$$

G.5 Datasets

G.5.1 Standard Poisson process

In-distribution sequences (corresponding to $\mathbb{P}_{\text{model}}$) are all generated from an SPP (i.e., a homogeneous Poisson process with rate $\mu = 1$) on the interval $[0, 100]$. The OoD sequences (corresponding to $\mathbb{Q}$) for each of the scenarios are generated as follows, where $\delta \in [0, 1]$ is the detectability parameter.

(i) Rate: homogeneous Poisson process with rate $\mu = 1 - 0.5\delta$.

(ii) Stopping: We generate a sequence $X = (t_1, \dots, t_N)$ from an SPP and then remove all the events $t_i \in [t_{\text{stop}}, T]$, where we compute $t_{\text{stop}} = T(1 - 0.3\delta)$.

(iii) Renewal: A renewal process, where the inter-event times τ_i are sampled i.i.d. from a Gamma distribution with shape $k = 1 - \delta$ and scale $\theta = \frac{1}{1-\delta}$. Thus, the expected inter-event time stays the same, but the variance of inter-event times increases for higher δ.

(iv) Hawkes: Hawkes process with conditional intensity $\lambda^*(t) = \mu + \alpha \sum_{t_j < t} \exp(-(t - t_j))$. The parameters are chosen as $\mu = 1 - \delta$ and $\alpha = \delta$.

(v) Inhomogeneous: inhomogeneous Poisson process with intensity $\lambda(t) = 1 + \beta \sin(\omega t)$, where $\omega = 2\pi/50$ and $\beta = 2\delta$.

(vi) SelfCorrecting: self-correcting process with intensity $\lambda^*(t) = \exp\left(\mu t - \sum_{t_j < t} \alpha\right)$, where we set $\mu = \delta + 10^{-5}$ and $\alpha = \delta$.

In all above scenarios setting $\delta = 0$ recovers the standard Poisson process, thus making $\mathbb{P}_{\text{data}}$ and $\mathbb{Q}$ indistinguishable. Note that the parameters in scenarios (iii)–(vi) are chosen such that the expected number of events N is always equal to T, like in the SPP. For all scenarios, $\mathcal{D}_{\text{train}}$, $\mathcal{D}^{\text{ID}}_{\text{test}}$ and $\mathcal{D}^{\text{OOD}}_{\text{test}}$ consist of 1000 sequences each.

Additional experiments. For completeness, we consider two more scenarios.

(vii) IncreasingRate: Similar to scenario (i), but now the rate is increasing instead as $\mu = 1 + 0.5\delta$.

(viii) RenewalB: Similar to scenario (iii), but the variance now decreases for higher δ. For this we define the parameters of the Gamma distribution as $k = \frac{1}{1-\delta}$ and $\theta = 1 - \delta$.

The results are shown in Figure G.1. As we can see, the same qualitative conclusions apply here as for the experiments in Section 6.5.1.

G.5.2 Simulated data

Server-Stop and Server-Overload: In-distribution sequences for both scenarios are generated by a multivariate Hawkes process with $C = 3$ marks on the interval $[0, 100]$ with following base rates $\boldsymbol{\mu}$ and influence matrix $\boldsymbol{A}$:

$$\boldsymbol{\mu} = \begin{pmatrix} 3 \\ 0 \\ 0 \end{pmatrix} \qquad \boldsymbol{A} = \begin{pmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \end{pmatrix}$$

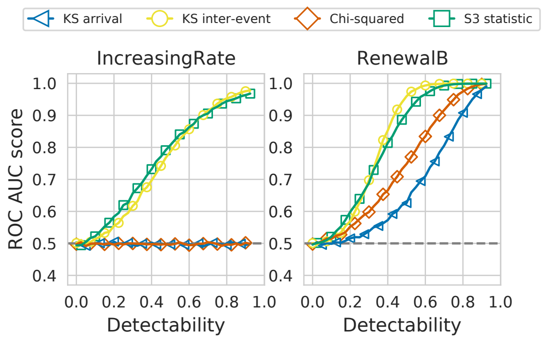


Figure G.1: GoF testing for the SPP using additional scenarios.

This scenario represents communication between a server (mark 1) and two worker machines (marks 2 and 3)—events for the workers can only be triggered by incoming requests from the server.

In OoD sequences, the structure of the influence matrix is changed at time $t_{\text{stop}} = T(1 - 0.5\delta)$, which represents the time of a failure in the system. For Server-Stop, the influence matrix is changed to $\boldsymbol{A}^{\text{stop}}$, and for Server-Overload the influence matrix is changed to $\boldsymbol{A}^{\text{overload}}$.

$$\boldsymbol{A}^{\text{stop}} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \end{pmatrix} \qquad \boldsymbol{A}^{\text{overload}} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 2 & 0 & 0 \end{pmatrix}$$

The sets $\mathcal{D}_{\text{train}}$, $\mathcal{D}_{\text{test}}^{\text{ID}}$ and $\mathcal{D}_{\text{test}}^{\text{OOD}}$ consist of 1000 sequences each.

Latency: Event sequences consist of two marks. ID sequences are generated as follows. Events of the first mark ("the trigger") are generated by a homogeneous Poisson process with rate $\mu = 3$. Events of the second mark ("the response") are obtained by shifting the arrival times of the first mark by offsets that are sampled i.i.d. from $\text{Normal}(\mu = 1, \sigma = 0.1)$. In OoD sequences, the offsets are instead sampled from $\text{Normal}(\mu = 1 + 0.5\delta, \sigma = 0.1)$. That is, OoD sequences correspond to increased latency between the "trigger" and "response" events. The sets $\mathcal{D}_{\text{train}}$, $\mathcal{D}_{\text{test}}^{\text{ID}}$ and $\mathcal{D}_{\text{test}}^{\text{OOD}}$ consist of 1000 sequences each.

SpikeTrains: The original fluorescence data is provided at `www.kaggle.com/c/connectomics`. We extracted the spike times from the fluorescence recordings using the code by `https://github.com/slinderman/pyhawkes/tree/master/data/chalearn`. We dequantized the discrete spike times by adding $\text{Uniform}(-0.5, 0.5)$ noise and selected the first 50 marks.

The original data consists of a single sequence that is 3590 seconds long. We split the long sequence into overlapping windows that are 20 seconds long. We select the first 500 sequences for training (as $\mathcal{D}_{\text{train}}$), and 96 remaining sequences for testing (as $\mathcal{D}_{\text{test}}^{\text{ID}}$). OoD sequences (i.e., $\mathcal{D}_{\text{test}}^{\text{OOD}}$) are obtained by switching $c = \lfloor \delta C \rfloor$ marks. For example, if marks 5 and 10 are switched, all events that correspond to mark 5 in $\mathcal{D}_{\text{test}}^{\text{ID}}$ will be labeled as mark 10 in $\mathcal{D}_{\text{test}}^{\text{OOD}}$, and vice versa.

G.5.3 Real-world data

LOGS: We ran the Sock Shop microservices testbed [190] on our in-house server. We consider the logs corresponding to the `user` service. There are 4 types of log entries that we model as 4 categorical marks. We use the timestamps of log entries as arrival times of a TPP. We slice the logs into 30-second-long non-overlapping windows, each corresponding to a single TPP realization.

We run the service for $\approx$14 hours to generate training data, and then for additional $\approx$5 hours to generate test data. The test data contains 5 types of injected anomalies produced by Pumba [104]. See Table 6.1 for the list of anomalies. Each anomaly injection lasts 10 minutes. We mark a test sequence as OoD if the system was "attacked" by Pumba during the respective time window. In total, we use 1668 sequences as $\mathcal{D}_{\text{train}}$, 502 sequences as $\mathcal{D}^{\text{ID}}_{\text{test}}$, and 22 sequences as $\mathcal{D}^{\text{OOD}}_{\text{test}}$ for each of the attack scenarios (i.e., 110 OoD sequences in total).

STEAD: The original dataset by Mousavi et al. [126] contains over 1 million earthquake recordings. We sample 72-hour sub-windows and treat times of earthquake as arrival times of a TPP, as usually done in seismological applications. We treat the sequences as unmarked. We select 4 geographic locations: (1) San Mateo, CA, (2) Anchorage, AK, (3) Aleutian Islands, AK, and (4) Hemet, CA. We group the earthquakes that happen within a 350 km radius (geodesic) around each of the locations, thus obtaining 4 sets of sequences (5000 sequences for each location). We use the sequences corresponding to (1) San Mateo, CA, as in-distribution data, and the remaining 3 locations as OoD data. We use 4000 ID sequences as $\mathcal{D}_{\text{train}}$, 1000 ID sequences and $\mathcal{D}^{\text{ID}}_{\text{test}}$, and 1000 sequences per each remaining location as $\mathcal{D}^{\text{OOD}}_{\text{test}}$.

G.6 Experimental setup

G.6.1 GoF for the SPP (Section 6.5.1)

We compute the p-values for the GoF test using the procedure described in Appendix G.4. For the GoF test, we use 1000 event sequences generated by an SPP as `samples` in the algorithm. The test statistics are computed using the compensator of the SPP $\Lambda^*(t) = t$. We compute the p-value for each event sequence in $\mathcal{D}^{\text{ID}}_{\text{test}}$ and $\mathcal{D}^{\text{OOD}}_{\text{test}}$, and then compute the ROC AUC score based on these p-values. The results are averaged over 10 random seeds.

G.6.2 OoD detection (Sections 6.5.2 & 6.5.3)

We train a neural TPP model similar to Shchur et al. [167]. We parametrize the inter-event time distribution with a mixture of 8 Weibull distributions. The marks are conditionally independent of the inter-event times given the context embedding, as in the original model. Mark embedding size is set to 32, and the context embedding (i.e., RNN hidden size) is set to 64 for all experiments.

We optimize the model parameters by maximizing the log-likelihood of the sequences in $\mathcal{D}_{\text{train}}$ (batch size 64) using Adam with learning rate 10^{-3} and clipping the L_2-norm of the gradients to 5. We run the optimization procedure for up to 200 epochs, and perform early stopping if the training loss stops improving for 10 epochs. The p-values are computed according to the procedure described in Appendix G.4. The results reported in Section 6.5.2 are averaged over 10 random seeds. In Section 6.5.3, we train the neural TPP model with 5 different random initializations to compute the average and standard error in Table 6.1.

G.7 Fisher's method for KS statistics

Here we show that ad-hoc fixes to the KS statistics that make them sensitive to the variations in the event count N lead to worse discriminative power in other scenarios. For this, we replicate the experimental setup from Section 6.5.1 with two additional statistics.

Fisher arrival. We compute the two-sided p-value p_N for the event count N using the CDF of the Poisson(V) distribution. Then, we compute the two-sided p-value $p_{\kappa_{\text{arr}}}$ for KS arrival statistic (Equation 6.5) using Kolmogorov distribution. We combine the two p-values using Fisher's method [63] as $-2(\log p_N + \log p_{\kappa_{\text{arr}}})$. **Fisher inter-event** is defined similarly using the two-sided p-value $p_{\kappa_{\text{int}}}$ for the KS inter-event statistic (Equation 6.6).

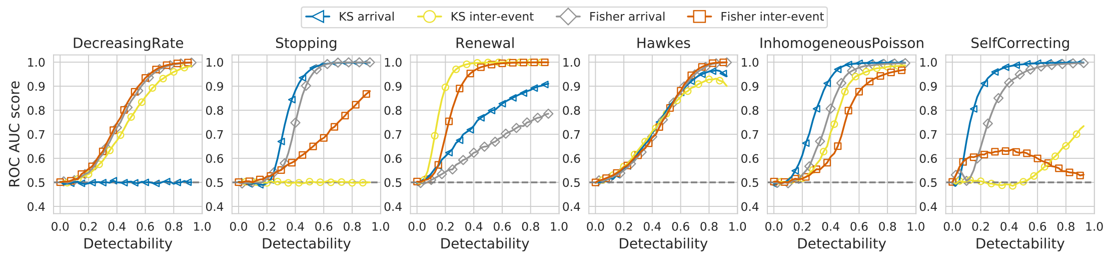


Figure G.2: Comparing KS statistics with the respective Fisher versions that are sensitive to the event count N.

Results. Results are shown in Figure G.2. We see that the Fisher versions of the statistics indeed fix the failure modes of the two KS scores on RATE and STOPPING, where the event count N changes in OoD sequences. However, the Fisher versions of the statistics perform worse than the respective KS statistics in 3 out of 4 remaining scenarios. In contrast, the 3S statistic performs well both in scenarios where N changes, as well as when the distribution of the arrival/inter-event times is varied.