Fast and Flexible Temporal Point Processes with Triangular Maps

Oleksandr Shchur, Nicholas Gao, Marin Biloš, Stephan Günnemann
Technical University of Munich, Germany
{shchur,gaoni,bilos,guennemann}@in.tum.de

Abstract

Temporal point process (TPP) models combined with recurrent neural networks provide a powerful framework for modeling continuous-time event data. While such models are flexible, they are inherently sequential and therefore cannot benefit from the parallelism of modern hardware. By exploiting the recent developments in the field of normalizing flows, we design TriTPP— a new class of non-recurrent TPP models, where both sampling and likelihood computation can be done in parallel. TriTPP matches the flexibility of RNN-based methods but permits orders of magnitude faster sampling. This enables us to use the new model for variational inference in continuous-time discrete-state systems. We demonstrate the advantages of the proposed framework on synthetic and real-world datasets.

1 Introduction

Temporal data lies at the heart of many high-impact machine learning applications. Electronic health records, financial transaction ledgers and server logs contain valuable information. A common challenge encountered in all these settings is that both the number of events and their times are variable. The framework of temporal point processes (TPP) allows us to naturally handle data that consists of variable-number events in continuous time. Du et al. [1] have shown that the flexibility of TPPs can be improved by combining them with recurrent neural networks (RNN). While such models are expressive and can achieve good results in various prediction tasks, they are poorly suited for sampling: sequential dependencies preclude parallelization. We show that it's possible to overcome the above limitation and design flexible TPP models without relying on RNNs. For this, we use the framework of triangular maps [2] and recent developments in the field of normalizing flows [3].

Our main contributions are: **(1)** We propose a new parametrization for several classic TPPs. This enables efficient parallel likelihood computation and sampling, which was impossible with existing parametrizations. **(2)** We propose TriTPP— a new class of non-recurrent TPPs. TriTPP matches the flexibility of RNN-based methods, while allowing orders of magnitude faster sampling. **(3)** We derive a differentiable relaxation for non-differentiable sampling-based TPP losses. This allows us to design a new variational inference scheme for Markov jump processes.

2 Background

Temporal point processes (TPP) [4] are stochastic processes that model the distribution of discrete events on some continuous time interval $[0, T]$. A realization of a TPP is a *variable-length* sequence of strictly increasing arrival times $\boldsymbol{t} = (t_1, \dots, t_N), t_i \in [0, T]$. We make the standard assumption and focus our discussion on regular finite TPPs [4]. One way to specify such a TPP is by using the (strictly positive) conditional intensity function $\lambda^*(t) := \lambda(t|\mathcal{H}_t)$ that defines the rate of arrival of new events

Code and datasets are available under `www.daml.in.tum.de/triangular-tpp`

34th Conference on Neural Information Processing Systems (NeurIPS 2020), Vancouver, Canada.

given the history $\mathcal{H}_t = \{t_j : t_j < t\}$. The $*$ symbol reminds us of the dependence on the history [5]. Equivalently, we can consider the *cumulative* conditional intensity $\Lambda^*(t) := \Lambda(t|\mathcal{H}_t) = \int_0^t \lambda^*(u)du$, also known as the compensator.[1] We can compute the likelihood of a realization $\boldsymbol{t}$ on $[0, T]$ as

$$p(\boldsymbol{t}) = \left(\prod_{i=1}^{N} \lambda^*(t_i)\right) \exp\left(-\int_0^T \lambda^*(u)du\right) = \left(\prod_{i=1}^{N} \frac{\partial}{\partial t_i}\Lambda^*(t_i)\right) \exp\left(-\Lambda^*(T)\right) \tag{1}$$

For example, we can use a TPP to model the online activity of a user in a 24-hour interval. In this case, each realization $\boldsymbol{t}$ could correspond to the timestamps of the posts by the user on a specific day.

Triangular maps [2] provide a framework that connects autoregressive models, normalizing flows and density estimation. Bogachev et al. [6] 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 $\boldsymbol{F} = (f_1, \ldots, f_N) : \mathbb{R}^N \to \mathbb{R}^N$ that pushes forward p into $\tilde{p}$.[2] A map $\boldsymbol{F}$ is called triangular if each component function f_i depends only on $(x_1, \ldots, x_i)$ 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

$$p(\boldsymbol{x}) = \left|\det J_{\boldsymbol{F}}(\boldsymbol{x})\right| \tilde{p}\left(\boldsymbol{F}(\boldsymbol{x})\right) = \left(\prod_{i=1}^{N} \frac{\partial}{\partial x_i} f_i(x_1, \ldots, x_i)\right) \tilde{p}\left(\boldsymbol{F}(\boldsymbol{x})\right) \tag{2}$$

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's 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).

3 Defining temporal point processes using triangular maps

We can notice the similarity between the right-hand sides of Equations 1 and 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 $\boldsymbol{t} = (t_1, \ldots, t_N)$ be a realization of a TPP on $[0, T]$ with compensator Λ^* (i.e. with density $p(\boldsymbol{t})$). The random time change theorem states that in this case $\boldsymbol{z} = (\Lambda^*(t_1), \ldots, \Lambda^*(t_N))$ is a realization of a homogeneous Poisson process (HPP) with unit rate on the interval $[0, \Lambda^*(T)]$ [4, Theorem 7.4.I][5, Proposition 4.1] (Figure 1).

The transformation $\boldsymbol{F} = (f_1, \ldots, f_N) : \boldsymbol{t} \mapsto \boldsymbol{z}$ is an increasing triangular map. Each component function $f_i(\boldsymbol{t}) = \Lambda(t_i|t_1, \ldots, t_{i-1})$ only depends on $(t_1, \ldots, t_i)$ 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 $\boldsymbol{t}$. Notice that the term $\prod_{i=1}^{N} \frac{\partial}{\partial t_i}\Lambda^*(t_i)$ in Equation 1 corresponds to the Jacobian determinant of $\boldsymbol{F}$. Similarly, the second term, $\tilde{p}(\boldsymbol{z}) = \tilde{p}(\boldsymbol{F}(\boldsymbol{t})) = \exp(-\Lambda^*(T))$, corresponds to the density of a HPP with unit rate on $[0, \Lambda^*(T)]$ for any realization $\boldsymbol{z}$. This demonstrates that all TPP densities (Equation 1) correspond to increasing triangular maps (Equation 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 C.3).

Our main idea is to define TPP densities $p(\boldsymbol{t})$ by directly specifying the respective maps $\boldsymbol{F}$. In Section 3.1, 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 3.2. Finally, we propose a new class of fast and flexible TPPs in Section 3.3.

3.1 Requirements for efficient TPP models

Density evaluation. The time complexity of computing the density $p(\boldsymbol{t})$ 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 $\boldsymbol{F}(\boldsymbol{t})$ takes $\mathcal{O}(N^2)$ operations. For example, this holds for Hawkes processes with

[1] For convenience, we provide a list of abbreviations and notation used in the paper in Appendix A.

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

arbitrary kernels [7]. If the compensator Λ^* has Markov property, the complexity of evaluating $\boldsymbol{F}$ can be reduced to $\mathcal{O}(N)$ *sequential* operations. This class of models includes Hawkes processes with exponential kernels [8, 9] and RNN-based autoregressive TPPs [1, 10, 11]. 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 $\mathcal{O}(N)$ *parallel* operations.

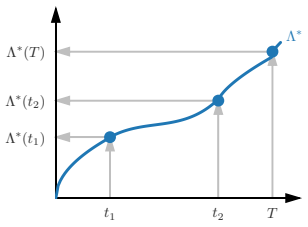


Figure 1: Triangular map $\boldsymbol{F}(\boldsymbol{t}) = (\Lambda^*(t_1), ..., \Lambda^*(t_N))$ is used for computing $p(\boldsymbol{t})$.

Sampling. As a converse of the random time change theorem, we can sample from a TPP density $p(\boldsymbol{t})$ by first drawing $\boldsymbol{z}$ from an HPP on $[0, \Lambda^*(T)]$ and applying the inverse map, $\boldsymbol{t} = \boldsymbol{F}^{-1}(\boldsymbol{z})$ [4]. There are, however, several caveats to this method. Not all parametrizations of $\boldsymbol{F}$ allow computing $\boldsymbol{F}^{-1}(\boldsymbol{z})$ in closed form. Even if $\boldsymbol{F}^{-1}$ is available, its evaluation for most models is again sequential [1, 9]. Lastly, the number of points N that will be generated (and thus $\Lambda^*(T)$ for HPP) is not known in advance. Therefore, existing methods typically resort to generating the samples one by one [5, Algorithm 4.1]. We show that it's 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 2). Even though this method may produce samples that are later discarded, it is much more efficient than sequential generation on GPUs (Section 6.1).

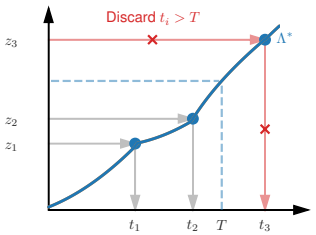


Figure 2: Sampling is done by applying $\boldsymbol{F}^{-1}$ to a sample $\boldsymbol{z}$ from a HPP with unit rate.

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 $\mathcal{O}(N)$ *parallel* operations. We will now show that maps corresponding to several classic TPP models can be defined to satisfy these criteria.

3.2 Fast temporal point process models

Inhomogeneous Poisson process (IPP) [4] is a TPP whose conditional intensity doesn't depend on the history, $\Lambda(t|\mathcal{H}_t) = \Lambda(t)$. The corresponding map is $\boldsymbol{F} = \boldsymbol{\Lambda}$, where $\boldsymbol{\Lambda}$ simply applies the function $\Lambda : [0, T] \to \mathbb{R}_+$ elementwise to the sequence $(t_1, ..., t_N)$.

Renewal process (RP) [12] 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 $\Phi : \mathbb{R}_+ \to \mathbb{R}_+$. The compensator of an RP is $\Lambda(t|\mathcal{H}_t) = \Phi(t - t_i) + \sum_{j=1}^{i} \Phi(t_j - t_{j-1})$, where t_i 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) [13] generalizes both inhomogeneous Poisson and renewal processes. The cumulative intensity is $\Lambda(t|\mathcal{H}_t) = \Phi(\Lambda(t) - \Lambda(t_i)) + \sum_{j=1}^{i} \Phi(\Lambda(t_j) - \Lambda(t_{j-1}))$. 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 $\mathcal{O}(N)$ 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 $\mathcal{O}(N)$ [14]. 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 [15]. Recently, Durkan et al. [3] used rational quadratic splines (RQS) to define functions that satisfy our requirements. We propose to use RQS to define Φ and Λ for (M)RP and IPP. This parametrization is flexible, while also allowing efficient density evaluation and sampling — something that existing approaches are unable to provide (see Section 5).

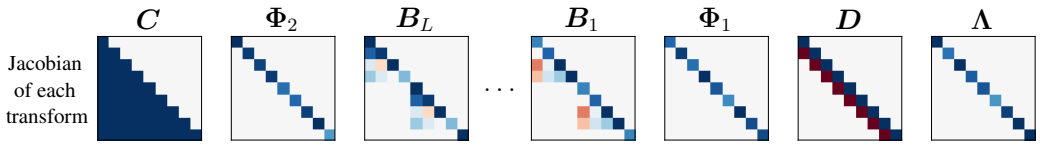


Figure 3: TriTPP defines an expressive map $\boldsymbol{F}$ as a composition of easy-to-invert transformations.

3.3 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 [7] or self-correcting [16] 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 $\mathcal{O}(N^2)$), 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 $\mathcal{O}(H^2)$, and multiplication by $\boldsymbol{B}_l$ or $\boldsymbol{B}_l^{-1}$ can be done in $\mathcal{O}(NH)$ 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 3 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 C).

4 Differentiable sampling-based losses for temporal point processes

Fast parallel sampling allows us to efficiently answer prediction queries such as "How many events are expected to happen in the next hour given the history?". More importantly, it enables us to efficiently train TPP models using objective functions of the form $\mathbb{E}_p[g(\boldsymbol{t})]$. This includes using $p(\boldsymbol{t})$ to specify the policy in reinforcement learning [17], to impute missing data during training [11] or to define an approximate posterior in variational inference (Section 4.2). In all but trivial cases the expression $\mathbb{E}_p[g(\boldsymbol{t})]$ has no closed form, so we need to estimate its gradients w.r.t. the parameters of $p(\boldsymbol{t})$ using Monte Carlo (MC). Recall that we can sample from $p(\boldsymbol{t})$ by applying the map $\boldsymbol{F}^{-1}$ to $\boldsymbol{z}$ drawn from an HPP with unit rate. This enables the so-called reparametrization trick [18]. Unfortunately, this is not enough. Sampling-based losses for TPPs are in general not differentiable. This is a property of the loss functions that is independent of the parametrization of $p(\boldsymbol{t})$ or the sampling method. In the following, we provide a simple example and a solution to this problem.

4.1 Entropy maximization

Consider the problem of maximizing the entropy of a TPP. An entropy penalty can be used as a regularizer during density estimation [19] or as a part of the ELBO in variational inference. Let $p_\lambda(\boldsymbol{t})$ be a homogeneous Poisson process on $[0, T]$ with rate $\lambda > 0$. It is known that the entropy is maximized when $\lambda = 1$ [20], but for sake of example assume that we want to learn λ that maximizes the entropy $-\mathbb{E}_p[\log p_\lambda(\boldsymbol{t})]$ with gradient ascent. We sample from $p_\lambda(\boldsymbol{t})$ by drawing a sequence $\boldsymbol{z} = (z_1, z_2, ...)$ from a HPP with unit rate and applying the inverse map $\boldsymbol{t} = \boldsymbol{F}_\lambda^{-1}(\boldsymbol{z}) = \frac{1}{\lambda}\boldsymbol{z}$

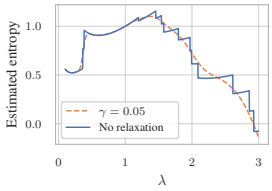


Figure 4: Monte Carlo estimate of the entropy.

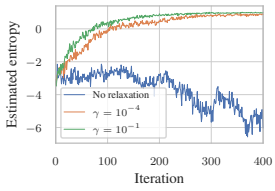


Figure 5: Maximizing the entropy with different values of γ.

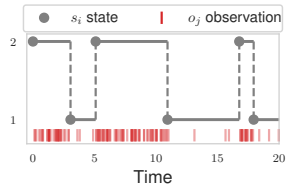


Figure 6: Markov modulated Poisson process with 2 states.

(Figure 2). We obtain an MC estimate of the entropy using a single such sample $\boldsymbol{t} = (t_1, t_2, ...)$ as

$$-\mathbb{E}_p[\log p_\lambda(\boldsymbol{t})] \approx \lambda T - \sum_{i=1}^{\infty} \mathbf{1}(t_i \leq T) \log \lambda = \lambda T - \sum_{i=1}^{\infty} \mathbf{1}\left(\frac{1}{\lambda} z_i \leq T\right) \log \lambda \tag{3}$$

Here, the indicator function $\mathbf{1}(\cdot)$ discards all the events $t_i > T$. We can see that for any sample $\boldsymbol{z}$ the right-hand side of Equation 3 is not continuous w.r.t. λ at points $\lambda = \frac{1}{T} z_i$. At such points, decreasing λ by an infinitesimal amount will "push" the sample $t_i = \frac{1}{\lambda} z_i$ outside the $[0, T]$ interval, thus increasing $\log p_\lambda(\boldsymbol{t})$ by a constant $\log \lambda$. We plot the right-hand side of Equation 3 as a function of λ in Figure 4, estimated with 5 MC samples. Clearly, such 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 $g(\boldsymbol{t})$ that depends on the events t_i inside $[0, T]$. For any TPP density $p(\boldsymbol{t})$, the discontinuities occur at the parameter values that map the HPP realizations z_i exactly to the interval boundary T.

Relaxation. We obtain a differentiable approximation to Equation 3 by relaxing the indicator functions as $\mathbf{1}(t_i \leq T) \approx \sigma_\gamma(T - t_i)$, where $\sigma_\gamma(x) = 1/(1 + \exp(-x/\gamma))$ is the sigmoid function with a temperature parameter $\gamma > 0$. Decreasing the temperature γ makes the approximation more accurate, but complicates optimization, similarly to the Gumbel-softmax trick [21]. Figure 5 shows convergence plots for different values of γ. Our relaxation applies to MC estimation of any function $g(\boldsymbol{t})$ that can be expressed in terms of the indicator functions. This method also enables differentiable sampling with reparametrization from a Poisson distribution, which might be of independent interest.

4.2 Variational inference for Markov jump processes

Combining fast sampling (Section 3) with the differentiable relaxation opens new applications for TPPs. As an example, we design a variational inference scheme for Markov jump processes.

Background. A Markov jump process (MJP) $\{s(t)\}_{t \geq 0}$ is a piecewise-constant stochastic process on $\mathbb{R}_+$. 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 $(\boldsymbol{t}, \boldsymbol{s})$ of jump times $\boldsymbol{t} = (t_1, ..., t_N)$ and the visited states $\boldsymbol{s} = (s_1, ..., s_{N+1})$. Note that N may vary for different trajectories. The prior over the trajectories $p(\boldsymbol{t}, \boldsymbol{s} | \boldsymbol{\pi}, \boldsymbol{A})$ of an MJP is governed by an initial state distribution $\boldsymbol{\pi}$ and a $K \times K$ generator matrix $\boldsymbol{A}$ (see Appendix B.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 $\boldsymbol{o}$. For concreteness, we consider the Markov-modulated Poisson process (MMPP) [22]. 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 6). Given the observations $\boldsymbol{o}$, we might be interested in inferring the trajectory $(\boldsymbol{t}, \boldsymbol{s})$, the model parameters $\boldsymbol{\theta} = \{\boldsymbol{\pi}, \boldsymbol{A}, \boldsymbol{\lambda}\}$, or both.

Variational inference. The posterior distribution $p(\boldsymbol{t}, \boldsymbol{s} | \boldsymbol{o}, \boldsymbol{\theta})$ of MMPP is intractable, so we approximate it with a variational distribution $q(\boldsymbol{t}, \boldsymbol{s}) = q(\boldsymbol{t}) q(\boldsymbol{s} | \boldsymbol{t})$. Note that this is *not* a mean-field approximation used in other works [23]. We model the distribution over the jump times $q(\boldsymbol{t})$ with TriTPP (Section 3.3). We find the best approximate posterior by maximizing the ELBO [24]

$$\max_{q(\boldsymbol{t})} \max_{q(\boldsymbol{s}|\boldsymbol{t})} \mathbb{E}_{q(\boldsymbol{t})} \left[\mathbb{E}_{q(\boldsymbol{s}|\boldsymbol{t})} \left[\log p(\boldsymbol{o} | \boldsymbol{t}, \boldsymbol{s}, \boldsymbol{\theta}) + \log p(\boldsymbol{t}, \boldsymbol{s} | \boldsymbol{\theta}) - \log q(\boldsymbol{t}, \boldsymbol{s}) \right] \right] \tag{4}$$

Given jump times $\boldsymbol{t}$, the true posterior over the states $p(\boldsymbol{s}|\boldsymbol{t}, \boldsymbol{o}, \boldsymbol{\theta})$ is just the posterior of a discrete hidden Markov model (HMM). This means that we only need to model $q(\boldsymbol{t})$; the optimal $q^\star(\boldsymbol{s}|\boldsymbol{t})$, i.e.

$$q^\star(\boldsymbol{s}|\boldsymbol{t}) = \arg\max_{q(\boldsymbol{s}|\boldsymbol{t})} \mathbb{E}_{q(\boldsymbol{s}|\boldsymbol{t})} \left[\log p(\boldsymbol{o}|\boldsymbol{t}, \boldsymbol{s}, \boldsymbol{\theta}) + \log p(\boldsymbol{t}, \boldsymbol{s}|\boldsymbol{\theta}) - \log q(\boldsymbol{s}|\boldsymbol{t})\right] = p(\boldsymbol{s}|\boldsymbol{t}, \boldsymbol{o}, \boldsymbol{\theta}) \quad (5)$$

can be found by doing inference in an HMM — doable efficiently via the forward-backward algorithm [25]. The inner expectation w.r.t. $q(\boldsymbol{s}|\boldsymbol{t})$ in Equation 4 can be computed analytically. We approximate the expectation w.r.t. $q(\boldsymbol{t})$ with Monte Carlo. Since all terms of Equation 4 are not differentiable, we apply our relaxation from Section 4.1. We provide a full derivation of the ELBO and the implementation details in Appendix B.2.

The proposed framework is not limited to approximating the posterior over the trajectories. With small modifications (Appendix B.3), we can simultaneously learn the parameters $\boldsymbol{\theta}$, either obtaining a point estimate $\boldsymbol{\theta}^\star$ or a full approximate posterior $q(\boldsymbol{\theta})$. Our variational inference scheme can also be extended to other continuous-time discrete-state models, such as semi-Markov processes [26].

5 Related work

Triangular maps [2] can be seen as a generalization of autoregressive normalizing flows [27, 28, 15]. Existing normalizing flow models are either limited to fixed-dimensional data [29, 30] or are inherently sequential [31, 32]. Our model proposed in Section 3.3 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 [33]. However, traditional inversion methods for IPPs are different from our approach (Section 3). 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 [34]. As an alternative to inversion, thinning approaches [35] 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, doesn't permit reparametrization, and is hard to express in terms of parallel operations on tensors [36]. Our inversion-based sampling addresses all the above limitations. It's 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 $p(t) = \lambda(t)/\Lambda(T)$ [37]. 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 [38]. Such models are flexible, but highly intractable. It's possible to devise approximations by, e.g., bounding the intensity function [39, 40]. 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 [24]. Recently, Morgan et al. [41] 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 [13, 42], but haven't 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 [43, 44]. Our proposed parametrization addresses the shortcomings of existing approaches and makes MRPs straightforward to apply in practice.

Neural TPPs. Du et al. [1] 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 [45], using more expressive conditional hazard functions [10, 46] or modeling the inter-event time distribution with normalizing flows [11]. All the above models are inherently sequential and therefore inefficient for sampling (Section 6.1). Recently, Turkmen et al. [36] 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.

Latent space models. TPPs governed by latent Markov dynamics have intractable likelihoods that require approximations [47, 48]. For MJPs, the state-of-the-art approach is the Gibbs sampler by Rao & Teh [49]. It allows to exactly sample from the posterior $p(\boldsymbol{t}, \boldsymbol{s}|\boldsymbol{o}, \boldsymbol{\theta})$, but is known to converge

slowly if the parameters $\boldsymbol{\theta}$ are to be learned as well [50]. Existing variational inference approaches for MJPs can only learn a fixed time discretization [23] or estimate the marginal statistics of the posterior [51, 52]. In contrast, our method (Section 4.2) produces a full distribution over the jump times.

6 Experiments

6.1 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 [53]. The architecture of the RNN model is nearly identical to the ones used in [1, 10, 11], except that the cumulative conditional hazard function is parametrized with a spline [3] to enable closed-form sampling. Appendix E 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 7 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 [54, 55] are defined in a similar 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.

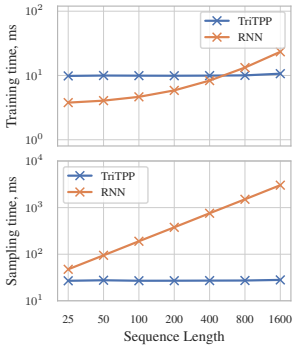


Figure 7: Scalability analysis. Standard devs. are below 1ms.

6.2 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 3.2 and Hawkes process [56].

Datasets. We use 6 synthetic datasets from Omi et al. [10]: Hawkes1&2 [7], self-correcting (SC) [16], 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 D for more details.

Metrics. The standard metric for comparing generative models, including TPPs, is negative log-likelihood (NLL) on a hold-out set [36, 10, 11]. 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 [57]. 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 [58]. For this reason, we additionally computed maximum mean discrepancy (MMD) [59] between the test sets and the samples drawn from each model after training. To measure similarity between two realizations $\boldsymbol{t}$ and $\boldsymbol{t}'$, we use a Gaussian kernel $k(\boldsymbol{t}, \boldsymbol{t}') = \exp(-d(\boldsymbol{t}, \boldsymbol{t}')/2\sigma^2)$, where $d(\boldsymbol{t}, \boldsymbol{t}')$ is the "counting measure" distance from [60,

Table 1: Average test set NLL on synthetic and real-world datasets (lower is better). Best NLL in **bold**, second best underlined. Results with standard deviations can be found in Appendix F.1.

	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 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**

Equation 3]. For completeness, we provide the definitions in Appendix E.2. MMD quantifies the dissimilarity between the true data distribution $p^\star(\boldsymbol{t})$ and the learned density $p(\boldsymbol{t})$ — lower is better.

Results. Table 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 [1]. 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, SC, 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 (SC, MRP, PUBG).

Table 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 F.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.

6.3 Variational inference

Setup. We apply our variational inference method (Section 4.2) for learning the posterior distribution over the latent trajectories of an MMPP. We simulate an MMPP with $K = 3$ latent states. As a baseline, we use the state-of-the-art MCMC sampler by Rao & Teh [49].

Results. Figure 8 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 two components that enable our new variational inference approach are our efficient parallel sampling algorithm for TriTPP (Section 3) and the differential relaxation (Section 4). Appendix F.2 contains an additional experiment on real-world data, where we both learn the parameters $\boldsymbol{\theta}$ and infer the posterior over the trajectories.

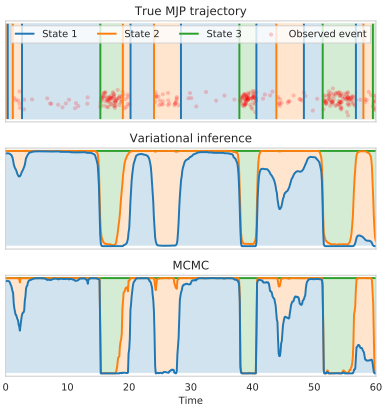


Figure 8: Posterior distributions over the latent trajectory of an MMPP learned using our VI approach & MCMC.

7 Future work & 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 [5]. 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 allow it to be used as a plug-and-play component of other machine learning models.

Broader impact

Existing works have applied TPPs and MJPs for analyzing electronic health records [61, 62], detecting anomalies in network traffic [63, 64] and modeling user behavior on online platforms [65, 66]. Thanks to fast sampling, our model can be used for solving new prediction tasks on such data, and the overall improved scalability allows practitioners to work with larger datasets. We do not find any of the above use cases ethically questionable, though, general precautions must be implemented when handling sensitive personal data. Since our model exploits fast parallel computations, has fewer parameters and converges in fewer iterations, it is likely to be more energy-efficient compared to RNN-based TPPs. However, we haven't performed experiments analyzing this specific aspect of our model.

Acknowledgments

This research was supported by the German Federal Ministry of Education and Research (BMBF), grant no. 01IS18036B, the Software Campus Project Deep-RENT and by the BMW AG. The authors of this work take full responsibilities for its content.

References

[1] 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.

[2] Priyank Jaini, Kira A Selby, and Yaoliang Yu. Sum-of-squares polynomial flow. *International Conference on Machine Learning*, 2019.

[3] Conor Durkan, Artur Bekasov, Iain Murray, and George Papamakarios. Neural spline flows. In *Advances in Neural Information Processing Systems*, 2019.

[4] Daryl J Daley and David Vere-Jones. An introduction to the theory of point processes. Vol. I. Probability and its applications, 2003.

[5] Jakob Gulddahl Rasmussen. Temporal point processes: the conditional intensity function. *Lecture Notes, Jan*, 2011.

[6] Vladimir Bogachev, Aleksandr Kolesnikov, and Kirill Medvedev. Triangular transformations of measures. *Sbornik: Mathematics*, 196(3):309, 2005.

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

[8] David Oakes. The Markovian self-exciting process. *Journal of Applied Probability*, 12(1):69–77, 1975.

[9] Angelos Dassios and Hongbiao Zhao. Exact simulation of Hawkes process with exponentially decaying intensity. *Electronic Communications in Probability*, 18, 2013.

[10] 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.

[11] Oleksandr Shchur, Marin Biloš, and Stephan Günnemann. Intensity-free learning of temporal point processes. *International Conference on Learning Representations*, 2020.

[12] David Roxbee Cox. *Renewal Theory*. Methuen, 1962.

[13] David Roxbee Cox. The statistical analysis of dependencies in point processes. *Stochastic Point Processes. Wiley: New York*, pages 55–66, 1972.

[14] Guy E Blelloch. Prefix sums and their applications. Technical report, 1990.

[15] George Papamakarios, Eric Nalisnick, Danilo Jimenez Rezende, Shakir Mohamed, and Balaji Lakshminarayanan. Normalizing flows for probabilistic modeling and inference. *arXiv preprint arXiv:1912.02762*, 2019.

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

[17] Utkarsh Upadhyay, Abir De, and Manuel Gomez Rodriguez. Deep reinforcement learning of marked temporal point processes. In *Advances in Neural Information Processing Systems*, 2018.

[18] Shakir Mohamed, Mihaela Rosca, Michael Figurnov, and Andriy Mnih. Monte Carlo gradient estimation in machine learning. *arXiv preprint arXiv:1906.10652*, 2019.

[19] Yves Grandvalet and Yoshua Bengio. Entropy regularization. *Semi-supervised learning*, pages 151–168, 2006.

[20] François Baccelli and Jae Oh Woo. On the entropy and mutual information of point processes. In *IEEE International Symposium on Information Theory*, 2016.

[21] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with Gumbel-softmax. *International Conference on Learning Representations*, 2017.

[22] Wolfgang Fischer and Kathleen Meier-Hellstern. The Markov-modulated Poisson process cookbook. *Performance Evaluation*, 18, 1993.

[23] Boqian Zhang, Jiangwei Pan, and Vinayak A Rao. Collapsed variational Bayes for Markov jump processes. In *Advances in Neural Information Processing Systems*, 2017.

[24] Cheng Zhang, Judith Bütepage, Hedvig Kjellström, and Stephan Mandt. Advances in variational inference. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 41(8):2008–2026, 2018.

[25] Pengyu Wang and Phil Blunsom. Collapsed variational Bayesian inference for hidden Markov models. In *Artificial Intelligence and Statistics*, 2013.

[26] William Feller. On semi-markov processes. *Proceedings of the National Academy of Sciences of the United States of America*, 51(4):653, 1964.

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

[28] 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.

[29] Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. Density estimation using Real NVP. *International Conference on Learning Representations*, 2017.

[30] George Papamakarios, Theo Pavlakou, and Iain Murray. Masked autoregressive flow for density estimation. In *Advances in Neural Information Processing Systems*, 2017.

[31] 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 arXiv:1609.03499*, 2016.

[32] Aaron Van den Oord, Nal Kalchbrenner, Lasse Espeholt, Oriol Vinyals, Alex Graves, et al. Conditional image generation with PixelCNN decoders. In *Advances in Neural Information Processing Systems*, 2016.

[33] Erhan Çinlar. *Introduction to Stochastic Processes*. Prentice-Hall, 1975.

[34] Raghu Pasupathy. Generating nonhomogeneous Poisson processes. *Wiley Encyclopedia of Operations Research and Management Science*, 2010.

[35] PA W Lewis and Gerald S Shedler. Simulation of nonhomogeneous Poisson processes by thinning. *Naval research logistics quarterly*, 26(3):403–413, 1979.

[36] Ali Caner Türkmen, Yuyang Wang, and Alexander J Smola. Fastpoint: Scalable deep point processes. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, 2019.

[37] David Roxbee Cox. The statistical analysis of series of events. *Monographs on Applied Probability and Statistics*, 1966.

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

[39] Ryan Prescott Adams, Iain Murray, and David JC MacKay. Tractable nonparametric Bayesian inference in Poisson processes with Gaussian process intensities. In *International Conference on Machine Learning*, 2009.

[40] Christian Donner and Manfred Opper. Efficient Bayesian inference of sigmoidal Gaussian Cox processes. *The Journal of Machine Learning Research*, 19, 2018.

[41] 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.

[42] Mark Berman. Inhomogeneous and modulated gamma processes. *Biometrika*, 68(1):143–152, 1981.

[43] Vinayak Rao and Yee W Teh. Gaussian process modulated renewal processes. In *Advances in Neural Information Processing Systems*, 2011.

[44] 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.

[45] Hongyuan Mei and Jason M Eisner. The neural Hawkes process: A neurally self-modulating multivariate point process. In *Advances in Neural Information Processing Systems*, 2017.

[46] Marin Biloš, Bertrand Charpentier, and Stephan Günnemann. Uncertainty on asynchronous time event prediction. In *Advances in Neural Information Processing Systems*, 2019.

[47] Marcel Hirt and Petros Dellaportas. Scalable bayesian learning for state space models using variational inference with smc samplers. *International Conference on Artificial Intelligence and Statistics*, 2019.

[48] Jing Wu, Owen Ward, James Curley, and Tian Zheng. Markov-modulated Hawkes processes for sporadic and bursty event occurrences. *arXiv preprint arXiv:1903.03223*, 2019.

[49] Vinayak Rao and Yee Whye Teh. Fast MCMC sampling for Markov jump processes and extensions. *The Journal of Machine Learning Research*, 14, 2013.

[50] Boqian Zhang and Vinayak Rao. Efficient parameter sampling for Markov jump processes. *arXiv preprint arXiv:1704.02369*, 2019.

[51] Manfred Opper and Guido Sanguinetti. Variational inference for Markov jump processes. In *Advances in Neural Information Processing Systems*, 2008.

[52] Christian Wildner and Heinz Koeppl. Moment-based variational inference for Markov jump processes. *International Conference on Machine Learning*, 2019.

[53] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*, 2019.

[54] Qiang Zhang, Aldo Lipani, Omer Kirnap, and Emine Yilmaz. Self-attentive hawkes process. International Conference on Machine Learning, 2020.

[55] Simiao Zuo, Haoming Jiang, Zichong Li, Tuo Zhao, and Hongyuan Zha. Transformer hawkes process. International Conference on Machine Learning, 2020.

[56] 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 arXiv:1707.03003*, 2017.

[57] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *International Conference on Learning Representations*, 2015.

[58] Lucas Theis, Aäron van den Oord, and Matthias Bethge. A note on the evaluation of generative models. *International Conference on Learning Representations*, 2016.

[59] Arthur Gretton, Karsten M Borgwardt, Malte J Rasch, Bernhard Schölkopf, and Alexander Smola. A kernel two-sample test. *Journal of Machine Learning Research*, 13, 2012.

[60] Shuai Xiao, Mehrdad Farajtabar, Xiaojing Ye, Junchi Yan, Le Song, and Hongyuan Zha. Wasserstein learning of deep generative point process models. In *Advances in Neural Information Processing Systems*, 2017.

[61] Ahmed M Alaa and Mihaela Van Der Schaar. A hidden absorbing semi-markov model for informatively censored temporal data: Learning and inference. *The Journal of Machine Learning Research*, 19(1):108–169, 2018.

[62] Qi Cao, Erik Buskens, Talitha Feenstra, Tiny Jaarsma, Hans Hillege, and Douwe Postmus. Continuous-time semi-markov models in health economic decision making: an illustrative example in heart failure disease management. *Medical Decision Making*, 36(1):59–71, 2016.

[63] Alexander Ihler, Jon Hutchins, and Padhraic Smyth. Learning to detect events with markov-modulated poisson processes. *ACM Transactions on Knowledge Discovery from Data*, 1(3), 2007.

[64] Ping Yan, Timothy Schoenharl, Alec Pawling, and Greg Madey. Anomaly detection in the WIPER system using Markov modulated Poisson process. 2007.

[65] Manuel Gomez Rodriguez, David Balduzzi, and Bernhard Schölkopf. Uncovering the temporal dynamics of diffusion networks. *International Conference on Machine Learning*, 2011.

[66] Charalampos Mavroforakis, Isabel Valera, and Manuel Gomez-Rodriguez. Modeling the dynamics of learning activity on the web. In *International Conference on World Wide Web*, 2017.

[67] Vinayak Rao and Yee W. Teh. MCMC for continuous-time discrete-state systems. In *Advances in Neural Information Processing Systems*, 2012.

A Abbreviations and notation

Abbreviations:

- RNN — recurrent neural network
- TPP — temporal point process
- HPP — homogeneous Poisson process (compensator is $\Lambda^*(t) = \lambda t$ for some $\lambda > 0$)
- IPP — inhomogeneous Poisson process
- RP — renewal process
- MRP — modulated renewal process
- MJP – Markov jump process
- MMPP — Markov modulated Poisson process
- MC — Monte Carlo
- VI — variational inference

Table 3: Notation used throughout the paper.

Notation	Description
$\boldsymbol{t} = (t_1, ..., t_N)$	Variable-length realization of a TPP.
$p(\boldsymbol{t})$	Density of a point process, also called likelihood (Equation 1).
$\lambda^*(t) = \lambda(t\|t_1, ..., t_{i-1})$	Conditional intensity at time t, where t_{i-1} is the last event before t.
$\Lambda^*(t) = \Lambda(t\|t_1, ..., t_{i-1})$ $= \int_0^T \lambda^*(u)du$	Cumulative conditional intensity at time t, also known as the compensator.
$\Lambda(t)$	(Unconditional) cumulative intensity of a Poisson process.
$\Phi(\tau)$	Cumulative hazard function of a renewal process.
$\boldsymbol{C}$	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, $D_{ij} = \begin{cases} 1 & \text{if } i = j, \\ -1 & \text{if } i = j + 1, \\ 0 & \text{else.} \end{cases}$
$\boldsymbol{F} = (f_1, ..., f_N)$	Increasing lower-triangular map that converts a realization $\boldsymbol{t}$ of an arbitrary TPP with compensator Λ^* into a sample $\boldsymbol{z}$ from an HPP with unit rate.
$f_i(t_1, ..., t_i) = \Lambda(t_i\|t_1, ..., t_{i-1})$	Component function of $\boldsymbol{F}$.
$\mathbf{1}(x)$	Indicator function, $\mathbf{1}(x) = \begin{cases} 1 & \text{if } x \text{ is True,} \\ 0 & \text{else.} \end{cases}$
γ	Temperature parameter for the diff. relaxation (Section 4.1).

B Variational inference for Markov jump processes

B.1 Generative model for MJP and MMPP

Markov jump process. We represent the trajectory of an MJP as a tuple $(\boldsymbol{t}, \boldsymbol{s})$, where $\boldsymbol{t} = (t_1, ..., t_N)$ are the (strictly increasing) jump times and $\boldsymbol{s} = (s_1, ..., s_{N+1})$ is the sequence of visited states. For convenience, we additionally set $t_0 = 0$ and $t_{N+1} = T$.

The distribution over the trajectories $(\boldsymbol{t}, \boldsymbol{s})$ is defined by a $K \times K$ generator matrix $\boldsymbol{A}$ and an initial state distribution $\boldsymbol{\pi}$. Each entry $A_{kl} \in \mathbb{R}_+$ denotes the rate of transition from state k to state l of the the MJP. Note that we use the formulation that permits self-jumps [67] (i.e., it may happen that $s_i = s_{i+1}$). We can 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

$$\begin{aligned} s_1 &\sim \text{Categorical}(\boldsymbol{\pi}) \\ t_i - t_{i-1} =: \tau_i &\sim \text{Exponential}\left(A_{s_i}\right) \\ s_{i+1} &\sim \text{Categorical}\left(\boldsymbol{A}_{s_i:}/A_{s_i}\right) \end{aligned} \tag{6}$$

Here, $\boldsymbol{A}_{s_i:}/A_{s_i}$ is the s_i'th row of $\boldsymbol{A}$ that is normalized to sum up to 1.

The likelihood of a trajectory $(\boldsymbol{t}, \boldsymbol{s})$ for an MJP with parameters $(\boldsymbol{\pi}, \boldsymbol{A})$ can be computed as

$$p(\boldsymbol{t}, \boldsymbol{s}|\boldsymbol{\pi}, \boldsymbol{A}) = \pi_{s_1} \left(\prod_{i=1}^{N} A_{s_{i-1} s_i} \right) \exp\left(- \sum_{i=1}^{N+1} (t_i - t_{i-1}) \sum_{l=1}^{K} A_{s_i l} \right)$$

We reformulate this expression using indicators $\mathbf{1}(\cdot)$, which will make the ELBO computation easier

$$= \left(\prod_{k=1}^{K} \pi_k^{\mathbf{1}(s_1=k)} \right) \left(\prod_{i=1}^{N} \prod_{k=1}^{K} \prod_{l=1}^{K} A_{kl}^{\mathbf{1}(s_i=k, s_{i+1}=l)} \right) \exp\left(- \sum_{i=1}^{N+1} (t_i - t_{i-1}) \sum_{k=1}^{K} \mathbf{1}(s_i = k) \left(\sum_{l=1}^{K} A_{kl} \right) \right)$$

By applying the logarithm to the above equation, we obtain

$$\begin{aligned} \log p(\boldsymbol{t}, \boldsymbol{s}|\boldsymbol{\pi}, \boldsymbol{A}) = &\left(\sum_{k=1}^{K} \mathbf{1}(s_1 = k) \log \pi_k \right) + \left(\sum_{i=1}^{N} \sum_{k=1}^{K} \sum_{l=1}^{K} \mathbf{1}(s_i = k, s_{i+1} = l) \log A_{kl} \right) \\ &- \left(\sum_{i=1}^{N+1} (t_i - t_{i-1}) \sum_{k=1}^{K} \mathbf{1}(s_i = k) \left(\sum_{l=1}^{K} A_{kl} \right) \right) \end{aligned} \tag{7}$$

Markov modulated Poisson process. Distribution of the observations $\boldsymbol{o} = (o_1, ..., o_M)$ of an MMPP depends on the latent MJP trajectory $(\boldsymbol{t}, \boldsymbol{s})$ and the rates of each state $\boldsymbol{\lambda} \in \mathbb{R}_+^K$. The observations $\boldsymbol{o}$ 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 $\boldsymbol{o}$ given $(\boldsymbol{t}, \boldsymbol{s})$ and $\boldsymbol{\lambda}$ can be computed as

$$p(\boldsymbol{o}|\boldsymbol{t}, \boldsymbol{s}, \boldsymbol{\lambda}) = \left(\prod_{i=1}^{N+1} \lambda_{s_i}^{M_{[t_{i-1}, t_i)}} \right) \exp\left(- \sum_{i=1}^{N+1} (t_i - t_{i-1}) \lambda_{s_i} \right)$$

where $M_{[t_{i-1}, t_i)}$ is the number of events o_j in the interval $[t_{i-1}, t_i)$. Again, using indicator functions, we rewrite it as

$$= \left(\prod_{i=1}^{N+1} \prod_{j=1}^{M} \left(\prod_{k=1}^{K} \lambda_k^{\mathbf{1}(s_i=k)} \right)^{\mathbf{1}(o_j \in [t_{i-1}, t_i))} \right) \exp\left(- \sum_{i=1}^{N+1} (t_i - t_{i-1}) \sum_{k=1}^{K} \mathbf{1}(s_i = k) \lambda_k \right)$$

By applying the logarithm, we obtain

$$\begin{aligned} \log p(\boldsymbol{o}|\boldsymbol{t}, \boldsymbol{s}, \boldsymbol{\lambda}) = &\left(\sum_{i=1}^{N+1} \sum_{j=1}^{M} \mathbf{1}(o_j \in [t_{i-1}, t_i) \sum_{k=1}^{K} \mathbf{1}(s_i = k) \log \lambda_k \right) \\ &- \left(\sum_{i=1}^{N+1} (t_i - t_{i-1}) \sum_{k=1}^{K} \mathbf{1}(s_i = k) \lambda_k \right) \end{aligned} \tag{8}$$

B.2 Derivation of the ELBO

The true posterior of an MMPP $p(\boldsymbol{t}, \boldsymbol{s}|\boldsymbol{o}, \boldsymbol{\pi}, \boldsymbol{A}, \boldsymbol{\lambda}) \propto p(\boldsymbol{t}, \boldsymbol{s}, \boldsymbol{o}|\boldsymbol{\pi}, \boldsymbol{A}, \boldsymbol{\lambda}) = p(\boldsymbol{t}, \boldsymbol{s}|\boldsymbol{\pi}, \boldsymbol{A})p(\boldsymbol{o}|\boldsymbol{t}, \boldsymbol{s}, \boldsymbol{\lambda})$ is intractable. We approximate it with a variational distribution $q(\boldsymbol{t}, \boldsymbol{s})$ by maximizing the evidence lower bound (ELBO) [24]

$$\mathrm{ELBO}(q, \boldsymbol{\theta}) = \mathbb{E}_{q(\boldsymbol{t},\boldsymbol{s})}\Bigg[\underbrace{\log p(\boldsymbol{t}, \boldsymbol{s}|\boldsymbol{\pi}, \boldsymbol{A})}_{\text{trajectory log-likelihood}} + \underbrace{\log p(\boldsymbol{o}|\boldsymbol{t}, \boldsymbol{s}, \boldsymbol{\lambda})}_{\text{observations log-likelihood}} \underbrace{- \log q(\boldsymbol{t}, \boldsymbol{s})}_{\text{entropy}}\Bigg]$$

Recall that we model the approximate posterior as $q(\boldsymbol{t}, \boldsymbol{s}) = q(\boldsymbol{t})q(\boldsymbol{s}|\boldsymbol{t})$, where $q(\boldsymbol{t})$ is defined using TriTPP and $q(\boldsymbol{s}|\boldsymbol{t})$ is evaluated exactly for each Monte Carlo sample $\boldsymbol{t}$. We rewrite the ELBO as

$$\mathrm{ELBO}(q, \boldsymbol{\theta}) = \mathbb{E}_{q(\boldsymbol{t})}\left[\mathbb{E}_{q(\boldsymbol{s}|\boldsymbol{t})}\left[\log p(\boldsymbol{t}, \boldsymbol{s}|\boldsymbol{\pi}, \boldsymbol{A}) + \log p(\boldsymbol{o}|\boldsymbol{t}, \boldsymbol{s}, \boldsymbol{\lambda}) - \log q(\boldsymbol{s}|\boldsymbol{t})\right] - \log q(\boldsymbol{t})\right]$$

We already derived the expressions for $\log p(\boldsymbol{t}, \boldsymbol{s}|\boldsymbol{\pi}, \boldsymbol{A})$ (Equation 7) and $\log p(\boldsymbol{o}|\boldsymbol{t}, \boldsymbol{s}, \boldsymbol{\lambda})$ (Equation 8). The expression for $\log q(\boldsymbol{s}|\boldsymbol{t})$ can be obtained similarly

$$\log q(\boldsymbol{s}|\boldsymbol{t}) = \sum_{i=1}^{N+1}\sum_{k=1}^{K} \mathbf{1}(s_i = k) \log q(s_i = k|\boldsymbol{t}) \tag{9}$$

Finally, to compute the log-density $\log q(\boldsymbol{t})$ of a single sample $\boldsymbol{t} = (t_1, ..., t_N)$ we use the procedure described in Appendix C. We denote $\boldsymbol{z} = (z_1, ..., z_N, z_{N+1}) = \boldsymbol{F}(t_1, ..., t_N, T)$, and z_{-1} is the last entry of $\boldsymbol{z}$.

$$\log q(\boldsymbol{t}) = \sum_{i=1}^{N} \log \left|\frac{\partial z_i}{\partial t_i}\right| - z_{-1} \tag{10}$$

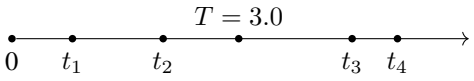
ELBO (non-differentiable version). Putting everything together, we get

$$\begin{aligned}
\mathrm{ELBO}(q, \boldsymbol{\theta}) = \mathbb{E}_{q(\boldsymbol{t})}\Bigg[&\mathbb{E}_{q(\boldsymbol{s}|\boldsymbol{t})}\Bigg[\sum_{k=1}^{K} \mathbf{1}(s_1 = k) \log \pi_k \\
&- \sum_{i=1}^{N+1} (t_i - t_{i-1}) \sum_{k=1}^{K} \mathbf{1}(s_i = k) \sum_{l=1}^{K} A_{kl} \\
&+ \sum_{i=1}^{N}\sum_{k=1}^{K}\sum_{l=1}^{K} \mathbf{1}(s_i = k, s_{i+1} = l) \log A_{kl} \\
&+ \sum_{i=1}^{N+1}\sum_{j=1}^{M} \mathbf{1}(o_j \in [t_i, t_{i+1})) \sum_{k=1}^{K} \mathbf{1}(s_i = k) \log \lambda_k \\
&- \sum_{i=1}^{N+1} (t_i - t_{i-1}) \sum_{k=1}^{K} \mathbf{1}(s_i = k) \lambda_k \\
&- \sum_{i=1}^{N+1}\sum_{k=1}^{K} \mathbf{1}(s_i = k) \log q(s_i = k|\boldsymbol{t})\Bigg] \\
&- \sum_{i=1}^{N} \log \left|\frac{\partial z_i}{\partial t_i}\right| + z_{-1}\Bigg]
\end{aligned} \tag{11}$$

Note that N is the length of the sample $\boldsymbol{t}$ generated from $q(\boldsymbol{t})$, so N will take different values for different samples. We can evaluate the inner expectation w.r.t. $q(\boldsymbol{s}|\boldsymbol{t})$ by using the following fact

$$\mathbb{E}_{q(\boldsymbol{s}|\boldsymbol{t})}\left[\mathbf{1}(s_i = k)\right] = q(s_i = k|\boldsymbol{t}) \quad \mathbb{E}_{q(\boldsymbol{s}|\boldsymbol{t})}\left[\mathbf{1}(s_i = k, s_{i+1} = l)\right] = q(s_i = k, s_{i+1} = l|\boldsymbol{t}) \tag{12}$$

Recall that we set $q(\boldsymbol{s}|\boldsymbol{t})$ to the true posterior $p(\boldsymbol{s}|\boldsymbol{t}, \boldsymbol{o}, \boldsymbol{\theta})$ over states given the jumps (Equation 5). This allows us to exactly compute both the posterior marginals $q(s_i = k|\boldsymbol{t})$ and the posterior



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	$\mathbf{1}(\boldsymbol{t} < T)$	$(1.00, 1.00, 0.00, 0.00)$
Relaxed indicator	$\sigma_\gamma(T - \boldsymbol{t})$	$(1.00, 0.97, 0.01, 0.00)$

Figure 9: Examples of values involved in the ELBO computation.

transition probabilities $q(s_i = k, s_{i+1} = l|\boldsymbol{t})$ using the forward-backward algorithm [25, Equations 7, 8]. Therefore, the inner expectation w.r.t. $q(\boldsymbol{s}|\boldsymbol{t})$ in Equation 11 can be computed analytically.

ELBO (differentiable relaxation). The ELBO, as defined above in Equation 11, is discontinuous w.r.t. the parameters of the density $q(\boldsymbol{t})$ for the reasons described in Section 4.1. The expression inside the expectation depends only on the events t_i that happen before T. Infinitesimal change in the parameters of $q(\boldsymbol{t})$ 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 C and Section 4.1. 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 C). We get a "clipped" / "padded" sample $\boldsymbol{t} = (t_1, ..., t_{N'})$ as $t_i = \min\{\tilde{t}_i, T\}$ (Figure 9). 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 $\boldsymbol{t}$:

$$
\begin{aligned}
\mathrm{ELBO}(q, \boldsymbol{\theta}) = \mathbb{E}_{q(\boldsymbol{t})} \Bigg[& \mathbb{E}_{q(\boldsymbol{s}|\boldsymbol{t})} \Bigg[\sum_{k=1}^{K} \mathbf{1}(s_1 = k) \log \pi_k \\
& - \sum_{i=1}^{N'} (t_i - t_{i-1}) \sum_{k=1}^{K} \mathbf{1}(s_i = k) \sum_{l=1}^{K} A_{kl} \\
& + \sum_{i=1}^{N'} \mathbf{1}(\tilde{t}_i < T) \sum_{k=1}^{K} \sum_{l=1}^{K} \mathbf{1}(s_i = k, s_{i+1} = l) \log A_{kl} \\
& + \sum_{i=1}^{N'} \sum_{j=1}^{M} \mathbf{1}(o_j \in [t_i, t_{i+1})) \sum_{k=1}^{K} \mathbf{1}(s_i = k) \log \lambda_k \\
& - \sum_{i=1}^{N'} (t_i - t_{i-1}) \sum_{k=1}^{K} \mathbf{1}(s_i = k) \lambda_k \\
& - \sum_{i=1}^{N'} \mathbf{1}(\tilde{t}_{i-1} < T) \sum_{k=1}^{K} \mathbf{1}(s_i = k) \log q(s_i = k|\boldsymbol{t}) \Bigg] \\
& - \sum_{i=1}^{N'} \mathbf{1}(\tilde{t}_i < T) \log \left| \frac{\partial z_i}{\partial t_i} \right| + z_{-1} \Bigg]
\end{aligned}
\tag{13}
$$

Changes from Equation 11 are highlighted in red. Even though the formula looks different, the result of evaluating Equation 13 will be *exactly* the same as for Equation 11. 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 $\mathbf{1}(t_i < T)$ with

sigmoids $\sigma_\gamma(T - t_i)$. The indicator function $\mathbf{1}(o_j \in [t_i, t_{i+1}))$ can also be relaxed as

$$\begin{aligned}\mathbf{1}(o_j \in [t_i, t_{i+1})) &= \mathbf{1}(t_{i+1} > o_j) - \mathbf{1}(t_i \geq o_j) \\ &\approx \sigma_\gamma(t_{i+1} - o_j) - \sigma_\gamma(t_i - o_j)\end{aligned} \tag{14}$$

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 $\{\boldsymbol{t}^{(1)}, \boldsymbol{t}^{(2)}, ...\}$ from $q(\boldsymbol{t})$, evaluate the posterior $q(\boldsymbol{s}|\boldsymbol{t})$ using with forward-backward for all of them in parallel, and evaluate the relaxed ELBO (Equation 13).

B.3 Parameter estimation

In Section 4.2, we perform approximate posterior inference over the trajectories $(\boldsymbol{t}, \boldsymbol{s})$ by maximizing the ELBO w.r.t. $q(\boldsymbol{t}, \boldsymbol{s})$

$$\max_{q(\boldsymbol{t},\boldsymbol{s})} \mathbb{E}_q[\log p(\boldsymbol{t}, \boldsymbol{s}|\boldsymbol{\theta}) + \log p(\boldsymbol{o}|\boldsymbol{t}, \boldsymbol{s}, \boldsymbol{\theta}) - \log q(\boldsymbol{t}, \boldsymbol{s})] \tag{15}$$

Since $\mathrm{ELBO}(q, \boldsymbol{\theta})$ provides a lower bound on the marginal log-likelihood $\log p(\boldsymbol{o}|\boldsymbol{\theta})$, we can also simultaneously learn the model parameters $\boldsymbol{\theta} = \{\boldsymbol{\pi}, \boldsymbol{A}, \boldsymbol{\lambda}\}$ by solving the following optimization problem (subject to appropriate constraints on $\boldsymbol{\theta}$)

$$\max_{\boldsymbol{\theta}} \max_{q(\boldsymbol{t},\boldsymbol{s})} \mathbb{E}_q[\log p(\boldsymbol{t}, \boldsymbol{s}|\boldsymbol{\theta}) + \log p(\boldsymbol{o}|\boldsymbol{t}, \boldsymbol{s}, \boldsymbol{\theta}) - \log q(\boldsymbol{t}, \boldsymbol{s})] \tag{16}$$

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 $p(\boldsymbol{\theta})$ and approximate $p(\boldsymbol{\theta}, \boldsymbol{t}, \boldsymbol{s}|\boldsymbol{x})$ with $q(\boldsymbol{\theta}, \boldsymbol{t}, \boldsymbol{s}) = q(\boldsymbol{\theta})q(\boldsymbol{t})q(\boldsymbol{s}|\boldsymbol{t}, \boldsymbol{\theta})$. This corresponds to the following optimization problem

$$\max_{q(\boldsymbol{\theta},\boldsymbol{t},\boldsymbol{s})} \mathbb{E}_q[\log p(\boldsymbol{t}, \boldsymbol{s}|\boldsymbol{\theta}) + \log p(\boldsymbol{o}|\boldsymbol{t}, \boldsymbol{s}, \boldsymbol{\theta}) - \log q(\boldsymbol{t}, \boldsymbol{s})] - \mathbb{KL}(q(\boldsymbol{\theta})\|p(\boldsymbol{\theta})) \tag{17}$$

where $\mathbb{KL}$ denotes KL-divergence. By applying our relaxation from Section 4, it's possible to solve all of the above optimization problems (Equations 15, 16, 17) using gradient ascent.

C Implementation details

C.1 Batch processing

By representing TPP densities with transformations, we can implement both (log-)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 $\boldsymbol{t}$ of different length.

First, consider a single sequence $\boldsymbol{t} = (1, 2.5, 4)$ with $N = 3$ events, sampled from a TPP on the interval $[0, 5]$. We pad this sequence with $T = 5$, and additionally introduce a mask $\boldsymbol{m}$ that tells us which entries of the padded vector $\boldsymbol{t}$ correspond to actual events (i.e., not padding)

$$\boldsymbol{t} = \begin{bmatrix}1 & 2.5 & 4 & 5\end{bmatrix} \qquad \boldsymbol{m} = \begin{bmatrix}1 & 1 & 1 & 0\end{bmatrix}$$

We implement the transformation $\boldsymbol{F}$ (corresponding to the TPP density $p(\boldsymbol{t})$) similarly to normalizing flow frameworks like `torch.distributions` [53]. 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 $\boldsymbol{F}$ (Table 3), 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

$$\log p(\boldsymbol{t}) = \texttt{sum}(\boldsymbol{m} \odot \boldsymbol{j}) - z_{-1} = \sum_{i=1}^{N'} m_i \log\left|\frac{\partial z_i}{\partial t_i}\right| - z_{-1} = \sum_{i=1}^{N} \log\left|\frac{\partial \Lambda^*(t_i)}{\partial t_i}\right| - \Lambda^*(T) \tag{18}$$

where N' denotes the length *with* the padding. We can verify that this is equal to the logarithm of the TPP density in Equation 1. Note that if we use a longer padding, such as

$$\boldsymbol{t} = \begin{bmatrix} 1 & 2.5 & 4 & 5 & 5 & 5 & 5 \end{bmatrix} \qquad \boldsymbol{m} = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 \end{bmatrix}$$

then Equation 18 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 3.3, 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.

C.2 Sampling

Sampling is implemented similarly. We start by simulating a vector $\tilde{\boldsymbol{z}}$ from a homogeneous Poisson process with unit rate. 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 $\boldsymbol{m}$ that can be obtained as $m_i = \mathbf{1}(\tilde{t}_i < T)$. In some use cases, such as entropy maximization (Section 4.1) or variational inference (Appendix B), we need to use a differentiable approximation to the mask $m_i = \sigma_\gamma(T - \tilde{t}_i)$. This recovers our relaxation from Section 4.1.

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 7, using larger sequence length leads to no noticeable computational overhead when using GPU.

C.3 Ensuring that the TPP is valid

We showed in Section 3 that every TPP density $p(\boldsymbol{t})$ 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 $\boldsymbol{z} = (z_1, ..., z_N)$ of arbitrary length N, there exists a unique increasing sequence $\boldsymbol{t} = (t_1, ..., t_N)$, such that $\boldsymbol{F}(\boldsymbol{t}) = \boldsymbol{z}$.

C.4 Parametrizing transformations using splines

Rational quadratic splines used by Durkan et al. [3] define a flexible nonlinear function $g : (0, 1) \to (0, 1)$. When defining our TPP models in Section 3, 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 C.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.

D Datasets

For each synthetic TPP model from Omi et al. [10, Section 4.1], we sampled 1000 sequences on the interval $[0, 100]$. This includes the **Hawkes1**, **Hawkes2**, **self-correcting (SC)**, **inhomogeneous Poisson (IPP)**, **modulated renewal (MRP)** and **renewal (RP)** processes.

PUBG.[3] 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.[4]

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[5] 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[6] contains the timestamps of the tweets by user 25073877, recorded over several years.

Yelp 1 and 2[7] contain the user check-in times for the McCarran International Airport and for all businesses in the city of Mississauga in 2018, respectively.

Table 4 shows the number of sequences, average sequence length and the duration of the $[0, T]$ interval for all the datasets.

E Experimental setup

E.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.

[3] `https://kaggle.com/skihikingkevin/pubg-match-deaths`

[4] `https://pushshift.io/`

[5] `https://www.kaggle.com/c/nyc-taxi-trip-duration/data`

[6] `https://twitter.com`

[7] `https://www.yelp.com/dataset/challenge`

Dataset name	Number of sequences	Average sequence length	Interval duration
Hawkes 1	1000	95.4	100
Hawkes 2	1000	97.2	100
SC	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

Table 4: Statistics for the synthetic & real-world datasets

E.2 Density estimation

NLL. In this experiment, we train all models by minimizing the average negative log-likelihood of the training set $\mathcal{D}_{\text{train}} = \{\boldsymbol{t}^{(1)}, \boldsymbol{t}^{(2)}, ...\}$

$$\min_{\boldsymbol{\theta}} -\frac{1}{|\mathcal{D}_{\text{train}}|} \frac{1}{N_{\text{avg}}} \sum_{\boldsymbol{t} \in \mathcal{D}_{\text{train}}} \log p_{\boldsymbol{\theta}}(\boldsymbol{t})$$

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 6.2 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 $p(\boldsymbol{t})$ learned by each model with the empirical distribution $p^\star(\boldsymbol{t})$ on the hold-out test set by estimating the maximum mean discrepancy (MMD) [59]. The MMD between distributions p and $p^\star$ is defined as

$$\text{MMD}(p, p^\star) = \mathbb{E}_{\boldsymbol{t},\boldsymbol{t}' \sim p}[k(\boldsymbol{t}, \boldsymbol{t}')] - 2\mathbb{E}_{\boldsymbol{t} \sim p, \boldsymbol{u} \sim p^\star}[k(\boldsymbol{t}, \boldsymbol{u})] + \mathbb{E}_{\boldsymbol{u},\boldsymbol{u}' \sim p^\star}[k(\boldsymbol{u}, \boldsymbol{u}')]$$

Here, $\boldsymbol{t} = (t_1, ..., t_N)$ and $\boldsymbol{u} = (u_1, ..., u_M)$ denote variable-length TPP realizations from different distributions, and $k(\cdot, \cdot)$ is a positive semi-definite kernel function that quantifies the similarity between two TPP realizations. We use the Gaussian kernel

$$k(\boldsymbol{t}, \boldsymbol{u}) = \exp\left(-\frac{d(\boldsymbol{t}, \boldsymbol{u})}{2\sigma^2}\right)$$

where $d(\boldsymbol{t}, \boldsymbol{u})$ is the counting measure distance between two TPP realizations from [60, Equation 3], defined as

$$d(\boldsymbol{t}, \boldsymbol{u}) = \sum_{i=1}^{N} |t_i - u_i| + \sum_{i=N+1}^{M} (T - u_i)$$

Here, we assume w.l.o.g. that $N \leq M$. Following Section 8 of Gretton et al. [59], the parameter σ is estimated as the median of $d(\boldsymbol{t}, \boldsymbol{u})$ with $\boldsymbol{t}, \boldsymbol{u} \sim p \cup p^\star$.

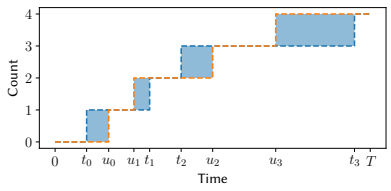


Figure 10: The blue area represents the counting measure distance (figure adapted from [60]).

E.3 Variational inference

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 $q(\boldsymbol{t})$ and use the temperature $\gamma = 0.1$ for the relaxation. We implemented the MCMC sampler by Rao & Teh [49] in Pytorch. We discard the first 100 samples (burn-in stage), and use 1000 samples to compute the marginal distribution of the posterior.

F Additional experiments

F.1 Density estimation

NLL table with standard deviations. For most models & datasets the results are nearly independent of the random initialization and the standard deviations are very close to zero. In the following table, we show the standard deviations of the NLL computed over 5 random initializations for all datasets where at least one of the models has the standard deviation above 0.005.

	PUBG		Reddit-S		Twitter		Yelp2	
	mean	std	mean	std	mean	std	mean	std
TriTPP	-2.41	0.34	-4.49	0.06	1.06	0.01	-0.09	0.01
RNN	-1.97	0.16	-4.89	0.29	1.08	0.01	-0.07	—
MRP	-0.83	0.08	-4.38	0.05	1.23	—	-0.1	—
RP	0.12	—	-4.01	0.03	1.2	—	-0.02	—
IPP	-0.06	—	-4.08	—	1.61	—	-0.05	—

Table 5: Average test set NLL with standard deviations. Datasets where all models have a standard deviation below 0.005 are excluded.

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 6 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 1).

Configuration	Hawkes1	Hawkes2	SC	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

Table 6: Test set NLL for different configurations of TriTPP.

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 11 and 12 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 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.

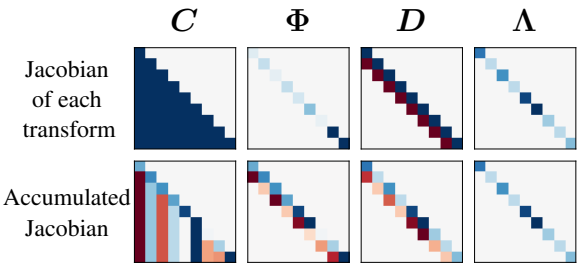


Figure 11: Jacobians of the component transformations of the modulated renewal process. We obtain the Jacobian of the combined transformation $\boldsymbol{F} = \boldsymbol{C} \circ \boldsymbol{\Phi} \circ \boldsymbol{D} \circ \boldsymbol{\Lambda}$ by multiplying the Jacobians of each transform (right to left).

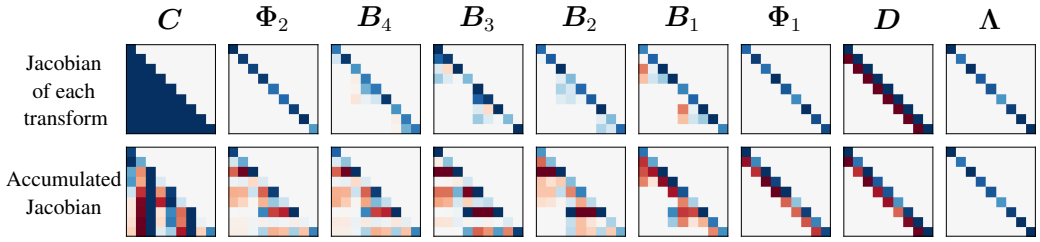
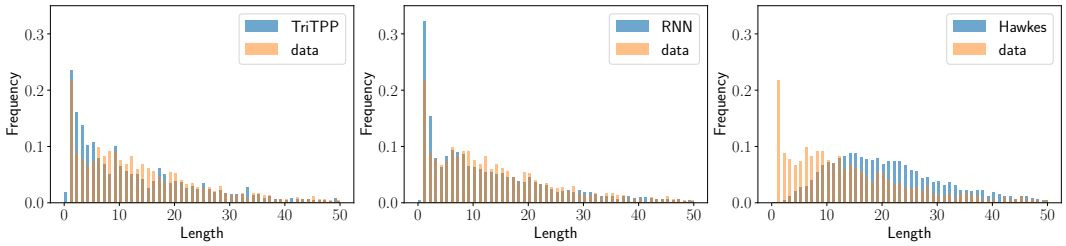


Figure 12: Jacobians of the component transformations of TriTPP. We obtain the Jacobian of the combined transformation $\boldsymbol{F} = \boldsymbol{C} \circ \boldsymbol{\Phi}_2 \circ \boldsymbol{B}_4 \circ \boldsymbol{B}_3 \circ \boldsymbol{B}_2 \circ \boldsymbol{B}_1 \circ \boldsymbol{\Phi}_1 \circ \boldsymbol{D} \circ \boldsymbol{\Lambda}$ 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 $\boldsymbol{t}$ 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 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 13 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 7. 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 1). However, when we sample sequences from the trained Hawkes model, the distribution of their lengths doesn't actually match the true data, as can be seen in Figure 13c.



(a) TriTPP (WD = 0.17) (b) RNN (WD = 0.10) (c) Hawkes (WD = 0.50)

Figure 13: Histograms of sequence lengths (true and generated) for Twitter. The difference between the two is quantified using Wasserstein distance (WD) — lower is better.

	Hawkes1	Hawkes2	SC	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**

Table 7: Wasserstein distance between the distributions of lengths of true and sampled sequences.

F.2 Variational inference

Random initliazations. In order to show that ours results are not cherry-picked, we provide the plots of marginal posterior trajectories (similar to Figure 8) obtained with 3 different random seeds. Figure 14 shows that our results are consistent across the random seeds.

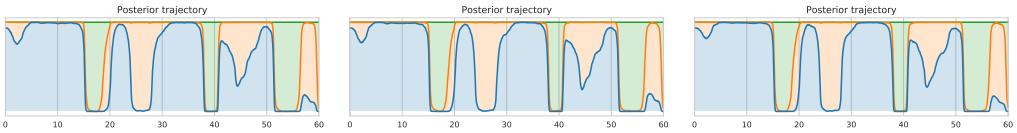


Figure 14: Marginal posterior trajectories obtained when using different random seeds.

Variational inference on real-world data. We apply our model to the server log data [8]. More specifically, we perform segmentation on the interval that contains the first 200 events. We estimate the posterior over the trajectories $(\boldsymbol{t}, \boldsymbol{s})$ and learn the model parameters $\boldsymbol{\theta} = \{\boldsymbol{\pi}, \boldsymbol{A}, \boldsymbol{\lambda}\}$ using the procedure described in Equation 16. Like before, we compare our approach to the MCMC sampler of Rao & Teh. For the MCMC sampler, we adopt an EM-like approach, where we alternate between closed-form parameter updates for $\boldsymbol{\theta}$ and simulating the posterior trajectories. Figure 15 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.

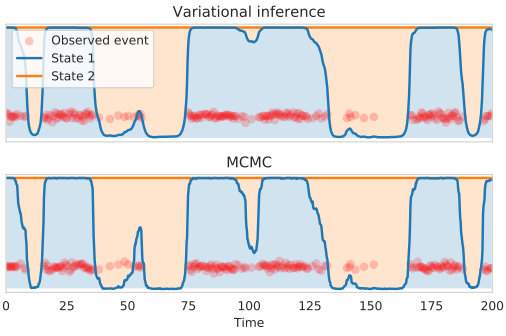
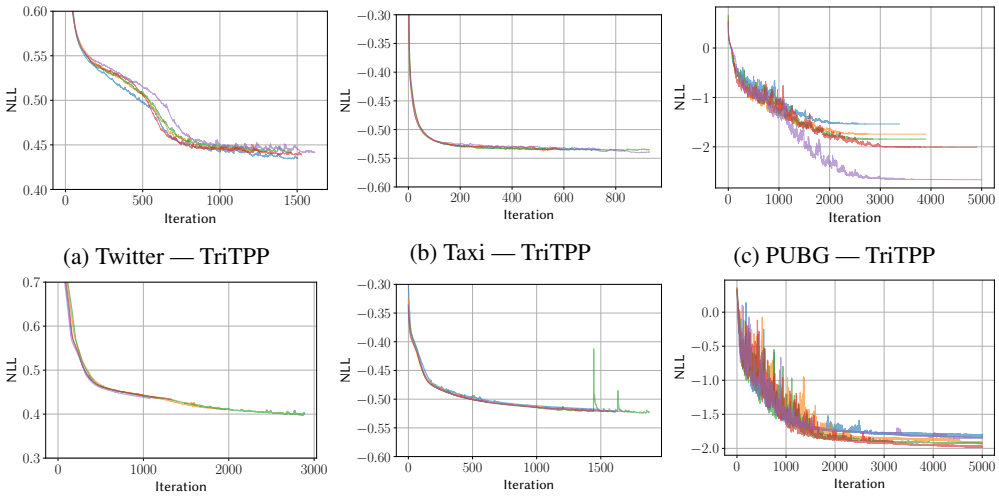


Figure 15: Segmentation of server data obtained using our VI approach and MCMC. In both cases, we estimate the posterior $p(\boldsymbol{t}, \boldsymbol{s}|\boldsymbol{o}, \boldsymbol{\theta})$ as well as the MMPP parameters $\boldsymbol{\theta}$.

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

F.3 Miscellaneous

Convergence plots for density estimation.



(a) Twitter — TriTPP (b) Taxi — TriTPP (c) PUBG — TriTPP

(d) Twitter — RNN (e) Taxi — RNN (f) PUBG — RNN

Figure 16: Training loss convergence for TriTPP and RNN model with different random seeds.

Convergence plots for variational inference.

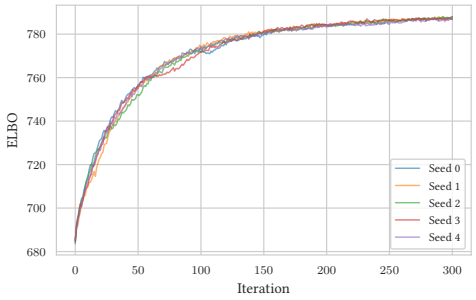


Figure 17: Convergence of our variational inference procedure when using 5 different random seeds.