Imputing Missing Events in Continuous-Time Event Streams

Hongyuan Mei [1] **Guanghui Qin** [2] **Jason Eisner** [1]

Abstract

Events in the world may be caused by other, *unobserved* events. We consider sequences of events in continuous time. Given a probability model of *complete* sequences, we propose particle smoothing—a form of sequential importance sampling—to impute the missing events in an *incomplete* sequence. We develop a trainable family of proposal distributions based on a type of bidirectional continuous-time LSTM. Bidirectionality lets the proposals condition on future observations, not just on the past as in particle filtering. Our method can sample an ensemble of possible complete sequences (particles), from which we form a single consensus prediction that has low Bayes risk under our chosen loss metric. We experiment in multiple synthetic and real domains, using different missingness mechanisms, and modeling the complete sequences in each domain with a neural Hawkes process (Mei & Eisner, 2017). On held-out incomplete sequences, our method is effective at inferring the ground-truth unobserved events, with particle smoothing consistently improving upon particle filtering.

1. Introduction

Event streams of discrete events in continuous time are often *partially* observed. We would like to impute the missing events $\mathbf{z}$. Suppose we know the prior distribution p_{model} of complete event streams, as well as the "missingness mechanism" $p_{\text{miss}}(\mathbf{z} \mid \text{complete stream})$, which stochastically determines which of the events will not be observed. One can then use use Bayes' Theorem, as spelled out in equation (1) below, to define the posterior distribution $p(\mathbf{z} \mid \mathbf{x})$ given just the observed events $\mathbf{x}$.[1]

Why is this important? The ability to impute $\mathbf{z}$ is useful in many applied domains, for example:

- *Medical records.* Some patients record detailed symptoms, self-administered medications, diet, and sleep. Imputing these events for other patients would produce an augmented medical record that could improve diagnosis, prognosis, treatment, and counseling. Similar remarks apply to users of life-tracking apps (e.g., MyFitnessPal) who forget to log some of their daily activities (e.g., meals, sleep and exercise).
- *Competitive games.* In poker or StarCraft, a player lacks full information about what her opponents have acquired (cards) or done (build mines and train soldiers). Accurately imputing hidden actions from "what I did" and "what I observed others doing" can help the player make good decisions. Similar remarks apply to practical scenarios (e.g., military) where multiple actors compete and/or cooperate.
- *User interface interactions.* Cognitive events are usually unobserved. For example, users of an online news provider (e.g., Bloomberg Terminal) may have read and remembered a displayed headline whether or not they clicked on it. Such events are expensive to observe (e.g., via gaze tracking or asking the user). Imputing them given the observed events (e.g., other clicks) would facilitate personalization.
- Other partially observed event streams arise in *online shopping*, *social media*, etc.

Why is it challenging? It is computationally difficult to reason about the posterior distribution $p(\mathbf{z} \mid \mathbf{x})$. Even for a simple p_{model} like a Hawkes process (Hawkes, 1971), Markov chain Monte Carlo (MCMC) methods are needed, and these methods obtain an efficient transition kernel only by exploiting special properties of the process (Shelton et al., 2018). Unfortunately, such properties no longer hold for the more flexible neural models that we will use in this paper (Du et al., 2016; Mei & Eisner, 2017).

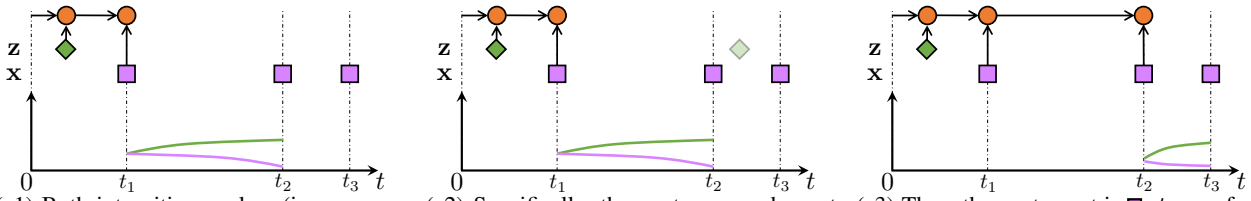
What is our contribution? We are, to the best of our knowledge, the first to develop general sequential Monte Carlo (SMC) methods to approximate the posterior distribution over incompletely observed draws from a neural point process. We begin by sketching the approach.

[1]Department of Computer Science, Johns Hopkins University, USA [2]Department of Physics, Peking University, China. Correspondence to: Hongyuan Mei <hmei@cs.jhu.edu>.

Proceedings of the 36^{th} International Conference on Machine Learning, Long Beach, California, PMLR 97, 2019.

[1]Bayes' Theorem can be applied even if p_{miss} is a missing-not-at-random (MNAR) mechanism, as is common in this setting. MNAR is only tricky if we know *neither* p_{model} *nor* p_{miss}.

Figure 1. Stochastically imputing a taxi's pick-up events (◆) given its observed drop-off events (■). At this stage, we are trying to determine the next event after the ■ at time t_1—either an unobserved event at $t_{1,1} \in (t_1, t_2)$ or the next observed event at t_2.

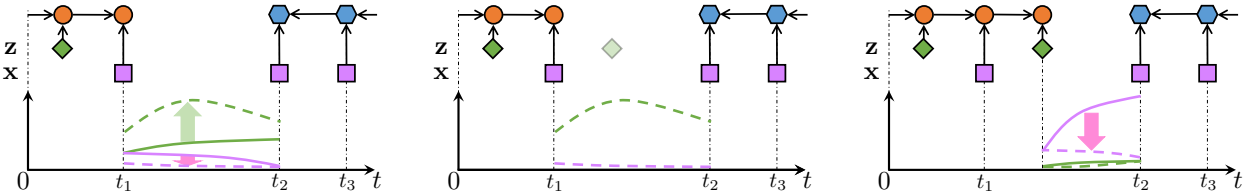


(a1) Both intensities are low (i.e., passengers are scarce at this time of day), so no event happens to be proposed in (t_1, t_2).

(a2) Specifically, the next proposed event (◇) would be *somewhere* after t_2, without bothering to determine its time precisely.

(a3) Thus, the next event is ■@t_2; we feed it into the LSTM, preempting ◇, which is discarded (line 37 of Algorithm 1).

(a) **Particle filtering (section 3.1)**. We show part of the process of drawing one particle. Above left, the neural Hawkes process's LSTM has already read the proposed and observed events at times $\leq t_1$. Its resulting state ● determines the model intensities ▬ and ▬ of the two event types ◆ and ■, from which the sampler (Algorithm 1 in Appendix C) determines that there is no unobserved event in (t_1, t_2). Above right, we continue to extend the particle by feeding ■@t_2 into the LSTM and proposing subsequent events based on the new intensities after t_2. But because ▬ was low at t_2, the ■@t_2 was unexpected, and that results in downweighting the particle (line 45 of Algorithm 1). Downweighting recognizes belatedly that proposing no event in (t_1, t_2) has committed us to a particle that will be improbable under the posterior, because its complete sequence includes consecutive drop-offs (■@t_1, ■@t_2) far apart in time.



(b1) Since a drop-off at t_2 strongly suggests a pick-up before t_2, considering the future increases the intensity of pick-up on (t_1, t_2) from ▬ to - - - (while decreasing that of drop-off from ▬ to - - -).

(b2) Consequently, the next proposed event is more likely to be a pick-up in (t_1, t_2) than it was in Figure 1a. If we stochastically generate such an event ◆@$t_{1,1}$, it is fed into the original ● LSTM.

(b3) The updated state ● determines the new *model* intensities ▬ and ▬, and also combines with ⬢ to determine the new *proposal* intensities - - - and - - -, which are used to sample the next event.

(b) **Particle smoothing (section 3.2)** samples from a better-informed proposal distribution: a second LSTM (Appendix D) reads the future observations from right to left, and its state ⬢ is used *together* with ● to determine the proposal intensities - - - and - - -.

Mei & Eisner (2017) give an algorithm to sample a complete sequence from a neural Hawkes process. Each event in turn is sampled given the complete history of previous events. However, this algorithm only samples from the prior over complete sequences. We first adapt it into a **particle filtering** algorithm that samples from the posterior given all the observed events. The basic idea (Figure 1a) is to draw the events in sequence as before, but now we *force* any observed events to be "drawn" at the appropriate times. That is, we add the observed events to the sequence as they happen (and they duly affect the distribution of subsequent events). There is an associated cost: if we are forced to draw an observed event that is *improbable* given its past history, we must downweight the resulting complete sequence accordingly, because evidently the particular past history that we sampled was inconsistent with the observed event, and hence cannot be part of a high-likelihood complete sequence. Using this method, we sample many sequences (or **particles**) of different *relative* weights. This method applies to any temporal point process.[2] Linderman et al. (2017) apply it to the classical Hawkes process.

Alas, this approach is computationally inefficient. Sampling a complete sequence that is actually probable under the posterior requires great luck, as the proposal distribution must have the good fortune to draw only events that happen to be consistent with future observations. Such lucky particles would appropriately get a high weight relative to other particles. The problem is that we will rarely get such particles at all (unless we sample very many).

To get a more accurate picture of the posterior, this paper draws each event from a smarter distribution that is conditioned on the future observations (rather than drawing the event in ignorance of the future and then downweighting the particle if the future does not turn out as hoped).

[2] As long as the number of events is finite with probability 1, and it is tractable to compute the log-likelihood of a complete sequence and to estimate the log-likelihoods of its prefixes.

This idea is called **particle smoothing** (Doucet & Johansen, 2009). How does it work in our setting? The neural Hawkes process defines the distribution of the next event using the state of a **continuous-time LSTM** that has read the past history from left to right. When sampling a proposed event, we now use a modified distribution (Figure 1b) that *also* considers the state of a second continuous-time LSTM that has read the future observations from right to left. As this modified distribution is still imperfect—merely a proposal distribution—we still have to reweight our particles to match the actual posterior under the model. But this reweighting is not as drastic as for particle filtering, because the new proposal distribution was constructed and trained to resemble the actual posterior. Our proposal distribution could also be used with other point process models by replacing the left-to-right LSTM state with other informative statistics of the past history.

What other contributions? We introduce an appropriate evaluation loss metric for event stream reconstruction, and then design a **consensus decoder** that outputs a single low-risk prediction of the missing events by combining the sampled particles (instead of picking one of them).

2. Preliminaries[3]

2.1. Partially Observed Event Streams

We consider a missing-data setting (Little & Rubin, 1987). We are given a fixed time interval $[0, T)$ over which events can be observed. An event of type $k \in \{1, 2, \ldots, K\}$ at time $t \in [0, T)$ is denoted by an ordered pair written mnemonically as $k@t$. Each possible outcome in our probability distributions is a complete event sequence in which each event is designated as either "observed" or "missing."

We observe only the **observed events**, denoted by $\mathbf{x} = \{k_1@t_1, k_2@t_2, \ldots, k_I@t_I\}$, where $0 = t_0 < t_1 < t_2 < \ldots < t_I < t_{I+1} = T$. We are given the observation interval $[0, T)$ in the form of two **boundary events** $k_0@t_0$ and $k_{I+1}@t_{I+1}$ at its endpoints, where $k_0 = 0$ and $k_{I+1} = K+1$.

Let $k_{i,0}@t_{i,0}$ be an alternative notation for the observed event $k_i@t_i$. Following this observed event (for any $0 \leq i \leq I$), there are $J_i \geq 0$ **unobserved events** $\mathbf{z} = \{k_{i,1}@t_{i,1}, k_{i,2}@t_{i,2}, \ldots, k_{i,J_i}@t_{i,J_i}\}$, where $t_{i,0} < t_{i,1} < \ldots < t_{i,J_i} < t_{i+1}$. We must guess this unobserved sequence including its length J_i. Let $\sqcup$ denote disjoint union. Our hypothesized **complete event sequence** $\mathbf{x} \sqcup \mathbf{z}$ is thus $\{k_{i,j}@t_{i,j} : 0 \leq i \leq I+1, 0 \leq j \leq J_i\}$, where $t_{i,j}$ increases strictly with the pair $\langle i, j \rangle$ in lexicographic order.[4]

[3] Our conventions regarding capitalization, boldface, etc. are inherited from the notation of Mei & Eisner (2017, section 2).

[4] In general we should allow $t_{i,j}$ to increase *non*-strictly with $\langle i, j \rangle$. But equality happens to have probability 0 under the neural Hawkes model. So it is convenient to exclude it here, simplifying notation by allowing $\mathbf{x}, \mathbf{z}, \mathcal{H}(t)$ to be sets, not sequences.

In this paper, we will attempt to guess all of $\mathbf{z}$ jointly by sampling it from the posterior distribution

$$p(Z = \mathbf{z} \mid X = \mathbf{x}) \propto p_{\text{model}}(Y = \mathbf{x} \sqcup \mathbf{z}) \cdot p_{\text{miss}}(Z = \mathbf{z} \mid Y = \mathbf{x} \sqcup \mathbf{z})$$

of a process that *first* generates the complete sequence $\mathbf{x} \sqcup \mathbf{z}$ from a complete data model p_{model} (given $[0, T)$), and *then* determines which events to censor with the possibly stochastic **missingness mechanism** p_{miss}. The random variables X, Z, and Y refer respectively to the sets of observed events, missing events, and all events over $[0, T)$. Thus $Y = X \sqcup Z$. Under the distributions we will consider, $|Y|$ is almost surely finite. Notice that $\mathbf{z}$ denotes the set of missing events in Y and $Z = \mathbf{z}$ denotes the fact that they are missing. That said, we will abbreviate our notation above in the standard way:

$$p(\mathbf{z} \mid \mathbf{x}) \;\propto\; p_{\text{model}}(\mathbf{x} \sqcup \mathbf{z}) \cdot p_{\text{miss}}(\mathbf{z} \mid \mathbf{x} \sqcup \mathbf{z}) \qquad (1)$$

Note that $\mathbf{x} \sqcup \mathbf{z}$ is simply an undifferentiated sequence of $k@t$ pairs; the subscripts $\langle i, j \rangle$ are in effect assigned by p_{miss}, which partitions $\mathbf{x} \sqcup \mathbf{z}$ into $\mathbf{x}$ and $\mathbf{z}$. To explain a sequence of 50 observed events, one hypothesis is that p_{model} generated 73 events and then p_{miss} selected 23 of them to be missing (as $\mathbf{z}$), leaving the 50 observed events (as $\mathbf{x}$).

In many missing data settings, the second factor of equation (1) can be ignored because (for the given $\mathbf{x}$) it is known to be a constant function of $\mathbf{z}$. Then the missing data are said to be **missing at random (MAR)**. For event streams, however, the second factor is generally not constant in $\mathbf{z}$ but varies with the *number* of missing events $|\mathbf{z}|$. Thus, our unobserved events are normally **missing not at random (MNAR)**. See discussion in section 5.1 and Appendix A.

2.2. Choice of p_{model}

We need a multivariate point process model $p_{\text{model}}(\mathbf{x} \sqcup \mathbf{z})$. We choose the **neural Hawkes process** (Mei & Eisner, 2017), which has proven flexible and effective at modeling many real-world event streams.

Whether an event happens at time $t \in [0, T)$ depends on the **history** $\mathcal{H}(t) \stackrel{\text{def}}{=} \{k'@t' \in \mathbf{x} \sqcup \mathbf{z} : t' < t\}$—the set of all *observed* and *unobserved* events before t. Given this history, the neural Hawkes process defines an **intensity** $\lambda_k(t \mid \mathcal{H}(t)) \in \mathbb{R}_{\geq 0}$, which may be thought of as the *instantaneous rate* at time t of events of type k:

$$\lambda_k(t \mid \mathcal{H}(t)) = f_k(\mathbf{v}_k^\top \mathbf{h}(t)) \qquad (2)$$

Here f_k is a softplus function with k-specific scaling parameter. The vector $\mathbf{h}(t) \in (-1, 1)^D$ summarizes $(\mathcal{H}(t), t)$. It is the hidden state at time t of a **continuous-time LSTM** that previously read the events in $\mathcal{H}(t)$ *as they happened*. The state of such an LSTM evolves endogenously as it waits between events, so the state $\mathbf{h}(t)$ reflects

not only the sequence of past events but also their *timing*, including the gap between the last event in $\mathcal{H}(t)$ and t.

As Mei & Eisner (2017) explain, the probability of an event of type k in the interval $[t, t+dt)$, divided by dt, approaches (2) as $dt \to 0^+$. Thus, λ_k is similar to the intensity function of an inhomogeneous Poisson process. Yet it is not a fixed parameter: the λ_k function for times $\geq t$ is affected by the previously sampled events $\mathcal{H}(t)$. See Appendix B.1.

3. Particle Methods

It is often intractable to sample *exactly* from $p(\mathbf{z} \mid \mathbf{x})$, because $\mathbf{x}$ and $\mathbf{z}$ can be interleaved with each other. As an alternative, we can use normalized importance sampling, drawing many $\mathbf{z}$ values from a **proposal distribution** $q(\mathbf{z} \mid \mathbf{x})$ and weighting them in proportion to $\frac{p(\mathbf{z}|\mathbf{x})}{q(\mathbf{z}|\mathbf{x})}$. Figure 1 shows the key ideas in terms of an example. Full details are spelled out in Algorithm 1 in Appendix C.

Algorithm 1 is a **Sequential Monte Carlo (SMC)** approach (Moral, 1997; Liu & Chen, 1998; Doucet et al., 2000; Doucet & Johansen, 2009). It returns an **ensemble of weighted particles** $\mathcal{Z}_M = \{(\mathbf{z}_m, w_m)\}_{m=1}^M$. Each particle $\mathbf{z}_m$ is sampled from the **proposal distribution** $q(\mathbf{z} \mid \mathbf{x})$, which is defined to support sampling via a *sequential* procedure that draws one unobserved event at a time. The corresponding w_m are **importance weights**, which are defined as follows (and built up factor-by-factor in Algorithm 1):

$$w_m \propto \frac{p_{\text{model}}(\mathbf{x} \sqcup \mathbf{z}_m)\; p_{\text{miss}}(\mathbf{z}_m \mid \mathbf{x} \sqcup \mathbf{z}_m)}{q(\mathbf{z}_m \mid \mathbf{x})} \geq 0 \qquad (3)$$

where the normalizing constant is chosen to make $\sum_{m=1}^M w_m = 1$. Equations (1) and (3) imply that we would have $w_m = 1/M$ if we could set $q(\mathbf{z} \mid \mathbf{x})$ equal to $p(\mathbf{z} \mid \mathbf{x})$, so that the particles were IID samples from the desired posterior distribution. In practice, q will not equal p, but will be easier than p to sample from. To correct for the mismatch, the importance weights w_m are higher for particles that q proposes less often than p would have proposed them.

The distribution implicitly formed by the ensemble, $\hat{p}(\mathbf{z})$, approaches $p(\mathbf{z} \mid \mathbf{x})$ as $M \to \infty$ (Doucet & Johansen, 2009). Thus, for large M, the ensemble may be used to estimate the expectation of *any* function $f(\mathbf{z})$, via

$$\mathbb{E}_{p(\mathbf{z}|\mathbf{x})}[f(\mathbf{z})] \approx \mathbb{E}_{\hat{p}}[f(\mathbf{z})] = \textstyle\sum_{m=1}^M w_m f(\mathbf{z}_m) \qquad (4)$$

$f(\mathbf{z})$ may be a function that summarizes properties of the complete stream $\mathbf{x} \sqcup \mathbf{z}$ on $[0, T)$, or predicts *future* events on $[T, \infty)$ using the sufficient statistic $\mathcal{H}(T) = \mathbf{x} \sqcup \mathbf{z}$.

In the subsections below, we will describe two specific proposal distributions q that are appropriate for the neural Hawkes process, as we sketched in section 1. These distributions define intensity functions λ^q over time intervals.

The trickiest part of Algorithm 1 (at line 31) is to sample the next unobserved event from the proposal distribution q. Here we use the **thinning algorithm** (Lewis & Shedler, 1979; Liniger, 2009; Mei & Eisner, 2017). Briefly, this is a rejection sampling algorithm whose own proposal distribution uses a *constant* intensity λ^*, making it a homogeneous Poisson process (which is easy to sample from). A event proposed by the Poisson process at time t is accepted with probability $\lambda^q(t)/\lambda^* \leq 1$. If it is rejected, we move on to the next event proposed by the Poisson process, continuing until we either accept such an unobserved event or are preempted by the arrival of the next observed event.

After each step, one may optionally *resample* a new set of particles from $\{\mathbf{z}_m\}_{m=1}^M$ (the RESAMPLE procedure in Algorithm 1). This trick tends to discard low-weight particles and clone high-weight particles, so that the algorithm can explore multiple continuations of the high-weight particles.

3.1. Particle Filtering

We already have a neural Hawkes process p_{model} that was trained on complete data. This model uses a neural net to define an intensity function $\lambda_k^p(t \mid \mathcal{H}(t))$ for *any* history $\mathcal{H}(t)$ of events before t and each event type k.

The simplest proposal distribution uses this intensity function to draw the unobserved events. More precisely, for each $i = 0, 1, \ldots, I$, for each $j = 0, 1, 2, \ldots$, let the next event $k_{i,j+1}@t_{i,j+1}$ be the first event generated by any of the K intensity functions $\lambda_k(t \mid \mathcal{H}(t))$ over the interval $t \in (t_{i,j}, t_{i+1})$, where $\mathcal{H}(t)$ consists of all observed and unobserved events up through $k_{i,j}@t_{i,j}$. If no event is generated on this interval, then the next event is $k_{i+1}@t_{i+1}$. This is implemented by Algorithm 1 with $smooth =$ **false**.

3.2. Particle Smoothing

As motivated in section 1, we would rather draw each unobserved event according to $\lambda_k(t \mid \mathcal{H}(t), \mathcal{F}(t))$ where the **future** $\mathcal{F}(t) \stackrel{\text{def}}{=} \{k_i@t_i : t < t_i \leq T\}$ consists of all *observed* events that happen after t. Note the asymmetry with $\mathcal{H}(t)$, which includes observed but also unobserved events.

We use a **right-to-left continuous-time LSTM** to summarize the future $\mathcal{F}(t)$ for any time t into another hidden state vector $\bar{\mathbf{h}}(t) \in (-1, 1)^{D'}$. Then we parameterize the proposal intensity using an extended variant of equation (2):

$$\lambda_k^q(t \mid \mathcal{H}(t), \mathcal{F}(t)) = f_k(\mathbf{v}_k^\top (\mathbf{h}(t) + \mathbf{B}\bar{\mathbf{h}}(t))) \qquad (5)$$

This extra machinery is used by Algorithm 1 when $smooth =$ **true**. Intuitively, the left-to-right $\mathbf{h}(t)$, as explained in Mei & Eisner (2017), reads the history $\mathcal{H}(t)$ and computes sufficient statistics for predicting events at times $\geq t$ given $\mathcal{H}(t)$. But we wish to predict these events given $\mathcal{H}(t)$ *and* $\mathcal{F}(t)$. Equation (5) approximates this Bayesian update using the right-to-left $\bar{\mathbf{h}}(t)$, which is trained to carry

back relevant information about future observations $\mathcal{F}(t)$.

This is a kind of neuralized forward-backward algorithm. Lin & Eisner (2018) treat the discrete-time analogue, explaining why a neural forward p_{model} no longer admits tractable exact proposals as does a hidden Markov model (Rabiner, 1989) or linear dynamical system (Rauch et al., 1965). Like them, we fall back on training an approximate proposal distribution. Regardless of p_{model}, particle smoothing is to particle filtering as Kalman smoothing is to Kalman filtering (Kalman, 1960; Kalman & Bucy, 1961).

Our right-to-left LSTM has the same architecture as the left-to-right LSTM used in our p_{model} (section 2.2), but a separate parameter vector. For any time $t \in [0, T)$, it arrives at $\overleftarrow{\mathbf{h}}(t)$ by reading *only* the *observed* events $\{k_i @ t_i : t < t_i \leq T\}$, i.e., $\mathcal{F}(t)$, in *reverse* chronological order. Formulas are given in Appendix D. This architecture seemed promising for reading an *incomplete* sequence of events from right to left, as Mei & Eisner (2017, section 6.3) had already found that this architecture is predictive when used to read incomplete sequences from left to right.

3.2.1. Training the Proposal Distribution

The particle smoothing proposer q can be trained to approximate $p(\mathbf{z} \mid \mathbf{x})$ by minimizing a **Kullback-Leibler (KL) divergence**. Its left-to-right LSTM is fixed at p_{model}, so its trainable parameters $\boldsymbol{\phi}$ are just the parameters of the right-to-left LSTM together with the matrix $\mathbf{B}$ from equation (5). Though $p(\mathbf{z} \mid \mathbf{x})$ is unknown, the gradient of **inclusive KL divergence** between $q(\mathbf{z} \mid \mathbf{x})$ and $p(\mathbf{z} \mid \mathbf{x})$ is

$$\nabla_\phi \mathrm{KL}(p \mid\mid q) = \mathbb{E}_{\mathbf{z} \sim p(\mathbf{z} \mid \mathbf{x})}[-\nabla_\phi \log q(\mathbf{z} \mid \mathbf{x})] \quad (6)$$

and the gradient of **exclusive KL divergence** is:

$$\nabla_\phi \mathrm{KL}(q \mid\mid p) = \mathbb{E}_{\mathbf{z} \sim q}[\nabla_\phi \left(\tfrac{1}{2} \left(\log q(\mathbf{z} \mid \mathbf{x}) - b\right)^2\right)] \quad (7a)$$

$$b = \log p_{\text{model}}(\mathbf{x} \sqcup \mathbf{z}) + \log p_{\text{miss}}(\mathbf{z} \mid \mathbf{x} \sqcup \mathbf{z}) \quad (7b)$$

where $\log p_{\text{model}}(\mathbf{x} \sqcup \mathbf{z})$ is given in Appendix B.1, $\log q(\mathbf{z} \mid \mathbf{x})$ is given in Appendix C.1, and $p_{\text{miss}}(\mathbf{z} \mid \mathbf{x} \sqcup \mathbf{z})$ is assumed to be known to us for any given pair of $\mathbf{x}$ and $\mathbf{z}$.

Minimizing inclusive KL divergence aims at high recall—$q(\mathbf{z} \mid \mathbf{x})$ is adjusted to assign high probabilities to all of the good hypotheses (according to $p(\mathbf{z} \mid \mathbf{x})$). Conversely, minimizing exclusive KL divergence aims at high precision—$q(\mathbf{z} \mid \mathbf{x})$ is adjusted to assign low probabilities to poor reconstructions, so that they will not be proposed. We seek to minimize the linearly combined divergence

$$\text{Div} = \beta\, \mathrm{KL}(p \| q) + (1 - \beta)\mathrm{KL}(q \| p) \text{ with } \beta \in [0, 1] \quad (8)$$

and training is early-stopped when the divergence stops decreasing on the held-out development set.

But how do we measure these divergences between $q(\mathbf{z} \mid \mathbf{x})$ and $p(\mathbf{z} \mid \mathbf{x})$? Of course, we actually want the *expected* divergence when the observed sequence $\mathbf{x} \sim$ the true distribution. Thus, we sample $\mathbf{x}$ by starting with a *fully observed* sequence from our training examples and then sampling a partition $\mathbf{x}, \mathbf{z}$ from the known missingness mechanism p_{miss}.[5] The inclusive expectation in (6) uses this $\mathbf{x}$ and $\mathbf{z}$. For the exclusive expectation in (7), we keep this $\mathbf{x}$ but sample a new $\mathbf{z}$ from our proposal distribution $q(\cdot \mid \mathbf{x})$.

Notice that minimizing exclusive divergence here is essentially the REINFORCE algorithm (Williams, 1992), which is known to have large variance. In practice, when tuning our hyperparameters (Appendix G.2), $\beta = 1$ in (8) gave the best results. That is—perhaps unsurprisingly—our experiments effectively avoided REINFORCE altogether and placed *all* the weight on the inclusive KL, which has no variance issue. More training details including a bias and variance discussion can be found in Appendix G.2.

Appendix H discusses situations where training on incomplete data by EM is possible.

4. A Loss Function and Decoding Method

It is often useful to find a *single* hypothesis $\hat{\mathbf{z}}$ that minimizes the *Bayes risk*, i.e., the expected loss with respect to the *unknown* ground truth $\mathbf{z}^*$. This procedure is called **minimum Bayes risk (MBR) decoding** and can be approximated with our ensemble of weighted particles:

$$\hat{\mathbf{z}} = \operatorname{argmin}_{\mathbf{z} \in \mathcal{Z}} \sum_{\mathbf{z}^* \in \mathcal{Z}} p(\mathbf{z}^* \mid \mathbf{x}) L(\mathbf{z}, \mathbf{z}^*) \quad (9a)$$

$$\approx \operatorname{argmin}_{\mathbf{z} \in \mathcal{Z}} \sum_{m=1}^{M} w_m L(\mathbf{z}, \mathbf{z}_m) \quad (9b)$$

where $L(\mathbf{z}, \mathbf{z}^*)$ is the **loss** of $\mathbf{z}$ with respect to $\mathbf{z}^*$. This procedure for combining the particles into a single prediction is sometimes called **consensus decoding**. We now propose a specific loss function L and an approximate decoder.

4.1. Optimal Transport Distance

The loss of $\mathbf{z}$ is defined as the minimum cost of editing $\mathbf{z}$ into the ground truth $\mathbf{z}^*$. To accomplish this edit, we must identify the best **alignment**—a one-to-one partial matching $\mathbf{a}$—of the events in the two sequences. We require any two aligned events to have the same type k. We define $\mathbf{a}$ as a collection of alignment edges (t, t^*) where t and t^* are the times of the aligned events in $\mathbf{z}$ and $\mathbf{z}^*$ respectively. An alignment edge between a predicted event at time t (in $\mathbf{z}$) and a true event at time t^* (in $\mathbf{z}^*$) incurs a cost of $|t - t^*|$ to move the former to the correct time. Each unaligned event in $\mathbf{z}$ incurs a deletion cost of C_{delete}, and each unaligned event in $\mathbf{z}^*$ incurs an insertion cost of C_{insert}. Now

$$L(\mathbf{z}, \mathbf{z}^*) = \min_{\mathbf{a} \in \mathcal{A}(\mathbf{z}, \mathbf{z}^*)} D(\mathbf{z}, \mathbf{z}^*, \mathbf{a}) \quad (10)$$

[5] To get more data for training q, we could sample more partitions of the fully observed sequence. In this paper, we only sample one partition. Note that the fully observed sequence is a real observation from the true complete data distribution (not the model).

where $\mathcal{A}(\mathbf{z}, \mathbf{z}^*)$ is the set of all possible alignments between $\mathbf{z}$ and $\mathbf{z}^*$, and $D(\mathbf{z}, \mathbf{z}^*, \mathbf{a})$ is the total cost given the alignment $\mathbf{a}$. Notice that if $|\mathbf{z}| \neq |\mathbf{z}^*|$, *any* alignment leaves some events unaligned; also, rather than align two faraway events, it is cheaper to leave them unaligned if $C_{\text{delete}} + C_{\text{insert}} < |t - t^*|$. Algorithm 2 in Appendix E uses dynamic programming to compute the loss (10) and its corresponding alignment $\mathbf{a}$, similar to edit distance (Levenshtein, 1965) or dynamic time warping (Sakoe & Chiba, 1971; Listgarten et al., 2005). In practice we symmetrize the loss by specifying equal costs $C_{\text{insert}} = C_{\text{delete}} = C$.

4.2. Consensus Decoding

Since aligned events must have the same type, consensus decoding (9b) decomposes into *separately* choosing a set $\hat{\mathbf{z}}^{(k)}$ of type-k events for *each* $k = 1, 2, \ldots, K$, based on the particles' sets $\mathbf{z}_m^{(k)}$ of type-k events. Thus, we simplify the presentation by omitting $^{(k)}$ throughout this section. The loss function L defined in section 4.1 warrants:

Theorem 1. *Given* $\{\mathbf{z}_m\}_{m=1}^M$, *if we define* $\mathbf{z}_\sqcup = \bigsqcup_{m=1}^M \mathbf{z}_m$, *then* $\exists \hat{\mathbf{z}} \subseteq \mathbf{z}_\sqcup$ *such that*

$$\textstyle\sum_{m=1}^M w_m L(\hat{\mathbf{z}}, \mathbf{z}_m) = \min_{\mathbf{z} \in \mathcal{Z}} \sum_{m=1}^M w_m L(\mathbf{z}, \mathbf{z}_m)$$

That is to say, there exists one subsequence of $\mathbf{z}_\sqcup$ *that achieves the minimum Bayes risk.*

The proof is given in Appendix F: it shows that if $\hat{\mathbf{z}}$ minimizes the Bayes risk but is *not* a subsequence of $\mathbf{z}_\sqcup$, then we can modify it to either improve its Bayes risk (a contradiction) or keep the same Bayes risk while making it a subsequence of $\mathbf{z}_\sqcup$ as desired.

Now we have reduced this decoding problem to a combinatorial optimization problem:

$$\hat{\mathbf{z}} = \operatorname{argmin}_{\mathbf{z} \subseteq \mathbf{z}_\sqcup} \textstyle\sum_{m=1}^M w_m L(\mathbf{z}, \mathbf{z}_m) \tag{11}$$

which is probably NP-hard, by analogy with the Steiner string problem (Gusfield, 1997).

Our heuristic (Algorithm 3 of Appendix F) seeks to iteratively improve $\hat{\mathbf{z}}$ by (1) using Algorithm 2 to find the optimal alignment $\mathbf{a}_m$ of $\hat{\mathbf{z}}$ with each $\mathbf{z}_m$, and then (2) repeating the following sequence of 3 phases until $\hat{\mathbf{z}}$ does not change. Each phase tries to update $\hat{\mathbf{z}}$ to decrease the weighted distance $\sum_{m=1}^M w_m D(\hat{\mathbf{z}}, \mathbf{z}_m, \mathbf{a}_m)$ which by Theorem 1 is an upper bound of the Bayes risk $\sum_{m=1}^M w_m L(\hat{\mathbf{z}}, \mathbf{z}_m)$:[6]

Move Phase For each event in $\hat{\mathbf{z}}$, move its time to the weighted median (using weights w_m) of the times of all $\leq M$ events that $\mathbf{a}_m$ aligns it to (if any), while keeping the alignment edges. This selects the new time that minimizes $\sum_{m=1}^M w_m D(\hat{\mathbf{z}}, \mathbf{z}_m, \mathbf{a}_m)$.

Delete Phase For each event in $\hat{\mathbf{z}}$, delete it (together with any related edges in each $\mathbf{a}_m$) if this decreases $\sum_{m=1}^M w_m D(\hat{\mathbf{z}}, \mathbf{z}_m, \mathbf{a}_m)$.

Insert Phase If we inserted t into $\hat{\mathbf{z}}$, we would also modify each $\mathbf{a}_m$ to align t to the closest unaligned event in $\mathbf{z}_m$ (if any) provided that this decreased $D(\hat{\mathbf{z}}, \mathbf{z}_m, \mathbf{a}_m)$. Let $\Delta(t)$ be the resulting reduction in $\sum_{m=1}^M w_m D(\hat{\mathbf{z}}, \mathbf{z}_m, \mathbf{a}_m)$. Let $t^* = \operatorname{argmax}_{t \in \mathbf{z}_\sqcup, t \notin \hat{\mathbf{z}}} \Delta(t)$. While $\Delta(t^*) > 0$, insert t^*.

The move or delete phase can consider events in any order, or in parallel; this does not change the result.

5. Experiments

We compare our particle smoothing method with the strong particle filtering baseline—our neural version of Linderman et al. (2017)'s Hawkes process particle filter—on multiple real-world and synthetic datasets. See Appendix G for training details (e.g., hyperparameter selection). PyTorch code can be found at `https://github.com/HMEIatJHU/neural-hawkes-particle-smoothing`.

5.1. Missing-Data Mechanisms

We experiment with missingness mechanisms of the form

$$p_{\text{miss}}(\mathbf{z} \mid \mathbf{x} \sqcup \mathbf{z}) = \prod_{k_i @ t_i \in \mathbf{z}} \rho_{k_i} \prod_{k_i @ t_i \in \mathbf{x}} (1 - \rho_{k_i}) \tag{12}$$

meaning that each event in the complete stream $\mathbf{x} \sqcup \mathbf{z}$ is independently censored with probability ρ_k that only depends on its event type k.[7] We consider both deterministic and stochastic missingness mechanisms. For the deterministic experiments, we set ρ_k for each k to be either 0 or 1, so that some event types are always observed while others are always missing. Then $p_{\text{miss}}(\mathbf{z} \mid \mathbf{x} \sqcup \mathbf{z}) = 1$ if $\mathbf{z}$ consists of precisely the events in $\mathbf{x} \sqcup \mathbf{z}$ that ought to go missing, and 0 otherwise. For our stochastic experiments, we simply set $\rho_k = \rho$ regardless of the event type k and experiment with $\rho = 0.1, 0.3, 0.5, 0.7, 0.9$. Then equation (12) can be written as $p_{\text{miss}}(\mathbf{z} \mid \mathbf{x} \sqcup \mathbf{z}) = (1 - \rho)^{|\mathbf{x}|} \rho^{|\mathbf{z}|}$, whose value decreases exponentially in the number of missing events $|\mathbf{z}|$. As this depends on $\mathbf{z}$, the stochastic setting is definitely MNAR (not MCAR as one might have imagined).

5.2. Datasets

The datasets that we use in this paper range from short sequences with mean length 15 to long ones with mean length > 300. For each of the datasets, we possess fully observed data that we use to train the model and the proposal distribution.[8] For each dev and test example, we censored

[6] Note these phases compute $D(\hat{\mathbf{z}}, \mathbf{z}_m, \mathbf{a}_m)$ but not $L(\hat{\mathbf{z}}, \mathbf{z}_m)$, so they need not call the dynamic programming algorithm.

[7] Appendix H discusses how $\boldsymbol{\rho}$ could be imputed when complete and incomplete data are both available.

[8] The focus of this paper is on inference (imputation) under a given model, so training the model is simply a preparatory step.

out some events from the fully observed sequence, so we present the $\mathbf{x}$ part as input to the proposal distribution but we also know the $\mathbf{z}$ part for evaluation purposes. Fully replicable details of the dataset preparation can be found in Appendix G, including how event types are defined and which event types are missing in the deterministic settings.

Synthetic Datasets We first checked that we could successfully impute unobserved events that are generated from *known* distributions. That is, when the generating distribution actually is a neural Hawkes process, could our method outperform particle filtering in practice? Is the performance consistent over multiple datasets drawn from different processes? To investigate this, we synthesized 10 datasets, each of which was drawn from a different neural Hawkes process with randomly sampled parameters.

Elevator System Dataset (Crites & Barto, 1996)**.** A multi-floor building is often equipped with multiple elevator cars that follow *cooperative* strategies to transport passengers between floors (Lewis, 1991; Bao et al., 1994; Crites & Barto, 1996). In this dataset, the events are which elevator car stops at which floor. The deterministic case of this domain is representative of many real-world cooperative (or competitive) scenarios—observing the activities of some players and imputing those of the others.

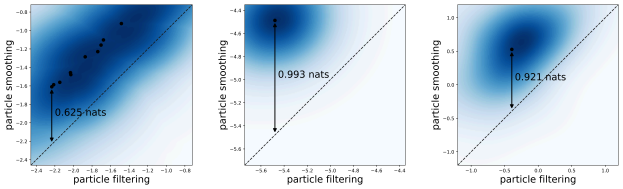
New York City Taxi Dataset (Whong, 2014)**.** Each medallion taxi in New York City has a sequence of time-stamped pick-up and drop-off events, where different locations have different event types. Figure 1 shows how we impute the pick-up events given the drop-off events (the deterministic missingness case).

5.3. Data Fitting Results

First, as an internal check, we measure *how probable* each ground truth reference $\mathbf{z}^*$ is under the proposal distribution constructed by each method, i.e., $\log q(\mathbf{z}^* \mid \mathbf{x})$. As shown in Figure 2, the improvement from particle smoothing is remarkably robust across 12 datasets, improving *nearly every* sequence in each dataset. The plots for the deterministic missingness mechanisms are so boringly similar that we only show them in Appendix G.6 (Figure 4).

5.4. Decoding Results

For each $\mathbf{x}$, we now make a prediction by sampling an ensemble of $M = 50$ particles (section 3)[9] and constructing their consensus sequence $\hat{\mathbf{z}}$ (section 4.2). We use multinomial resampling since otherwise the effective sample size



(a) Synthetic Data (b) Elevator System (c) NYC Taxi

Figure 2. Scatterplots of neural Hawkes particle smoothing (y-axis) vs. particle filtering (x-axis) with a stochastic missingness mechanism ($\rho = 0.5$). Each point represents a single test sequence, and compares the values of $\log q(\mathbf{z}^* \mid \mathbf{x}) \;/\; |\mathbf{z}^*|$. Larger values mean that the proposal distribution is better at proposing the ground truth $\mathbf{z}^*$. Each dataset's scatterplot is converted to a cloud using kernel density estimation, with the centroid denoted by a black dot. A double-arrowed line indicates the improvement of particle smoothing over filtering. For the synthetic datasets, we draw ten clouds on the same figure and show the line for the dataset where smoothing improves the most. As we can see, the density is always well concentrated above $y = x$. That is, this is not merely an average improvement: *nearly every* ground truth $\mathbf{z}^*$ gets higher proposal probability! Particle smoothing performs well even on datasets where particle filtering performs badly.

is very low (only 1–2 on some datasets).[10] We evaluate $\hat{\mathbf{z}}$ by its optimal transport distance (section 4.1) to the ground truth $\mathbf{z}^*$. Note that $\forall \mathbf{a}$, we can decompose $D(\hat{\mathbf{z}}, \mathbf{z}^*, \mathbf{a})$ as

$$C \cdot \big(\underbrace{|\hat{\mathbf{z}}| + |\mathbf{z}^*| - 2|\mathbf{a}|}_{\text{total insertions+deletions}} \big) + \underbrace{\textstyle\sum_{(t,t^*)\in\mathbf{a}} |t - t^*|}_{\text{total distance moved}} \qquad (13)$$

Letting $\mathbf{a}$ be the alignment that minimizes $D(\hat{\mathbf{z}}, \mathbf{z}^*, \mathbf{a})$, the former term measures how well $\hat{\mathbf{z}}$ predicts *which* events happened, and the latter measures how well $\hat{\mathbf{z}}$ predicts *when* those events happened. Different choices of C yield different $\hat{\mathbf{z}}$ with different trade-offs between these two terms. Intuitively, when $C \approx 0$, the decoder is free to insert and delete event tokens; as C increases, $\hat{\mathbf{z}}$ will tend to insert/delete fewer event tokens and move more of them.

Figure 3 plots the performance of particle smoothing (▲) vs. particle filtering (●) for the stochastic missingness mechanisms, showing the two terms above as the x and y coordinates. The very similar plots for the deterministic missingness mechanisms are in Appendix G.6 (Figure 5).[11]

5.5. Sensitivity to Missingness Mechanism

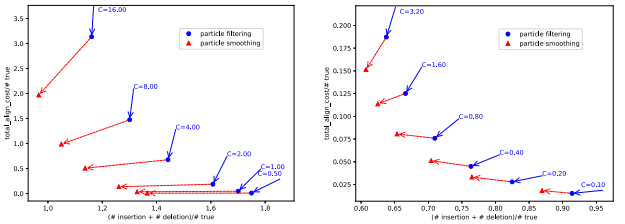
For the stochastic missingness mechanisms, we also did experiments with different values of missing rate $\rho = 0.1, 0.3, 0.7, 0.9$. Our particle smoothing method consistently outperforms the filtering baseline in all the experiments (Figure 6 in Appendix G.7), similar to Figure 3.

However, inference could be used to help train on incomplete data via the EM algorithm, provided that the missingness mechanism is known; see Appendix H for discussion.

[9] Increasing M would increase both effective sample size (ESS) and runtime.

[10] Any multinomial resampling step drives the ESS metric to M. This cannot guarantee better samples in general, but resampling did improve our decoding performance on all datasets.

[11] We show the 2 real datasets only. The figures for the 10 synthetic datasets are boringly similar to these.



(a) Elevator System (b) NYC Taxi

Figure 3. Optimal transport distance of particle smoothing (▲) vs. particle filtering (●) on test data with a stochastic missingness mechanism ($\rho = 0.5$). In each figure, the x-axis is the total number of deletions and insertions in the test dataset, $\sum_{n=1}^{N}(|\hat{\mathbf{z}}_n| + |\mathbf{z}^*{}_n| - 2|\mathbf{a}_n|)$, and the y-axis is the total movement cost, $\sum_{n=1}^{N}\sum_{(t,t^*)\in\mathbf{a}_n}|t-t^*|$. Both axes are normalized by the true total number of missing events $\sum_{n=1}^{N}|\mathbf{z}^*{}_n|$, so the x-axis shows a fraction and the y-axis shows an average time difference. On each dataset, we show one ● per C. According to equation (13), $(C, 1)$, denoted by ⟵, turns out to be the *gradient* of $\sum_{n=1}^{N} D(\hat{\mathbf{z}}_n, \mathbf{z}^*{}_n, \mathbf{a}_n)$ at this ●. The <- - shows the actual improvement obtained by switching to particle smoothing (which is, indeed, an improvement because it has positive dot product with the gradient ⟵). The Pareto frontier (convex hull) of the ▲ symbols dominates the Pareto frontier of the ● symbols—lying everywhere to its left—which means that our particle smoothing method outperforms the filtering baseline.

5.6. Runtime

The theoretical runtime complexity is $O(MI)$ where M is the number of particles and I is the number of observed events. In practice, we generate the particles in parallel, leading to acceptable speeds of 300-400 milliseconds per event for the final method. More details about the wall-clock runtime can be found in Appendix G.8.

6. Discussion and Related Work

To our knowledge, this is the first time a bidirectional recurrent neural network has been extended to predict events in continuous time. Bidirectional architectures have proven effective at predicting linguistic words and their properties given their left *and right* contexts (Graves et al., 2013; Bahdanau et al., 2015; Peters et al., 2018; Devlin et al., 2018): in particular, Lin & Eisner (2018) recently applied them to particle smoothing for discrete-time sequence tagging.

Previous work that infers unobserved events in continuous time exploits special properties of simpler models, including Markov jump processes (Rao & Teh, 2012; 2013), continuous-time Bayesian networks (Fan et al., 2010) and Hawkes processes (Shelton et al., 2018). Such properties no longer hold for our more expressive neural model, necessitating our approximate inference method.

Metropolis-Hastings would be an alternative to our particle smoothing method. The transition kernel could propose a single-event change to $\mathbf{z}$ (insert, delete, or move). Unfortunately, this would be quite slow for a neural model like ours, because any proposed change early in the sequence would affect the LSTM state and hence the probability of all subsequent events. Thus, a single move takes $O(|\mathbf{x} \sqcup \mathbf{z}|)$ time to evaluate. Furthermore, the Markov chain may mix slowly because a move that changes only one event may often lead to an incoherent sequence that will be rejected. The point of our particle smoothing is essentially to avoid such rejection by proposing a *coherent sequence of events*, left to right but considering future $\mathbf{x}$ events, from an approximation $q(\mathbf{z} \mid \mathbf{x})$ to the true posterior. (One might build a better Metropolis-Hastings algorithm by designing a transition kernel that makes use of our current proposal distribution, e.g., via particle Gibbs (Chopin & Singh, 2015).)

We also introduced an optimal transport distance between event sequences, which is a valid metric. It essentially regards each event sequence as a 0-1 function over times, and applies a variant of Wasserstein distance (Villani, 2008) or Earth Mover's distance (Kantorovitch, 1958; Levina & Bickel, 2001). Such variants are under active investigation (Benamou, 2003; Chizat et al., 2015; Frogner et al., 2015; Chizat et al., 2018). Our version allows event insertion and deletion during alignment, where these operations can only apply to an entire event—we cannot align half of an event and delete the other half. Due to these constraints, dynamic programming rather than a linear programming relaxation is needed to find the optimal transport. Xiao et al. (2017) also proposed an optimal transport distance between event sequences that allows event insertion and deletion; however, their insertion and deletion costs turn out to depend on the timing of the events in (we feel) a peculiar way.

We also gave a method to find a single "consensus" reconstruction with small average distance to our particles. This problem is related to Steiner string (Gusfield, 1997), which is usually reduced to multiple sequence alignment (MSA) (Mount, 2004) and heuristically solved by progressive alignment construction using a guide tree (Feng & Doolittle, 1987; Larkin et al., 2007; Notredame et al., 2000) and iterative realignment of the initial sequences with addition of new sequences to the growing MSA (Hirosawa et al., 1995; Gotoh, 1996). These methods might also be tried in our setting. For us, however, the i^{th} event of type k is not simply a character in a finite alphabet such as $\{\texttt{A}, \texttt{C}, \texttt{G}, \texttt{T}\}$ but a time that falls in the infinite set $[0, T)$. The substitution cost between two events of type k is then their time difference.

On multiple synthetic and real-world datasets, our method turns out to be effective at inferring the ground truth sequence of unobserved events. The improvement of particle smoothing upon particle filtering is substantial and consistent, showing the benefit of training a proposal distribution.

Acknowledgments

We are grateful to Bloomberg L.P. for enabling this work through a Ph.D. Fellowship Award to the first author. The work was also supported by an Amazon Research Award and by the National Science Foundation under Grant No. 1718846. We thank the anonymous reviewers, Hongteng Xu at Duke University, and our lab group at Johns Hopkins University's Center for Language and Speech Processing for helpful comments. We also thank NVIDIA Corporation for kindly donating two Titan X Pascal GPUs and the state of Maryland for the Maryland Advanced Research Computing Center. The first version of this work appeared on OpenReview in September 2018.

References

Bahdanau, D., Cho, K., and Bengio, Y. Neural machine translation by jointly learning to align and translate. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2015.

Bao, G., Cassandras, C. G., Djaferis, T. E., Gandhi, A. D., and Looze, D. P. Elevators dispatchers for down-peak traffic, 1994.

Baran, I., Demaine, E. D., and Katz, D. A. Optimally adaptive integration of univariate lipschitz functions. *Algorithmica*, 50(2):255–278, February 2008. URL https://link.springer.com/article/10.1007/s00453-007-9093-7.

Benamou, J.-D. Numerical resolution of an unbalanced mass transport problem. *ESAIM: Mathematical Modelling and Numerical Analysis*, 2003.

Chizat, L., Peyré, G., Schmitzer, B., and Vialard, F.-X. Unbalanced optimal transport: Geometry and Kantorovich formulation. *arXiv preprint arXiv:1508.05216*, 2015.

Chizat, L., Peyré, G., Schmitzer, B., and Vialard, F.-X. An interpolating distance between optimal transport and Fisher-Rao metrics. *Foundations of Computational Mathematics*, 2018.

Chopin, N. and Singh, S. S. On particle Gibbs sampling. *Bernoulli*, 2015.

Crites, R. H. and Barto, A. G. Improving elevator performance using reinforcement learning. In *Advances in Neural Information Processing Systems*, 1996.

Dempster, A. P., Laird, N. M., and Rubin, D. B. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, 1977.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. BERT: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

Doucet, A. and Johansen, A. M. A tutorial on particle filtering and smoothing: Fifteen years later. *Handbook of Nonlinear Filtering*, 2009.

Doucet, A., Godsill, S., and Andrieu, C. On sequential Monte Carlo sampling methods for Bayesian filtering. *Statistics and Computing*, 2000.

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

Fan, Y., Xu, J., and Shelton, C. R. Importance sampling for continuous-time Bayesian networks. *Journal of Machine Learning Research*, 2010.

Feng, D.-F. and Doolittle, R. F. Progressive sequence alignment as a prerequisite to correct phylogenetic trees. *Journal of Molecular Evolution*, 1987.

Frogner, C., Zhang, C., Mobahi, H., Araya, M., and Poggio, T. A. Learning with a Wasserstein loss. In *Advances in Neural Information Processing Systems*, 2015.

Gotoh, O. Significant improvement in accuracy of multiple protein sequence alignments by iterative refinement as assessed by reference to structural alignments. *Journal of Molecular Biology*, 1996.

Graves, A., Jaitly, N., and Mohamed, A.-R. Hybrid speech recognition with deep bidirectional LSTM. In *IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU)*, 2013.

Gusfield, D. *Algorithms on Strings, Trees and Sequences: Computer Science and Computational Biology*. 1997.

Hawkes, A. G. Spectra of some self-exciting and mutually exciting point processes. *Biometrika*, 1971.

Hirosawa, M., Totoki, Y., Hoshida, M., and Ishikawa, M. Comprehensive study on iterative algorithms of multiple sequence alignment. *Bioinformatics*, 1995.

Hochreiter, S. and Schmidhuber, J. Long short-term memory. *Neural Computation*, 1997.

Huang, Z., Xu, W., and Yu, K. Bidirectional LSTM-CRF models for sequence tagging. *arXiv preprint arXiv:1508.01991*, 2015.

Kalman, R. E. A new approach to linear filtering and prediction problems. *Journal of Basic Engineering*, 1960.

Kalman, R. E. and Bucy, R. S. New results in linear filtering and prediction theory. *Journal of Basic Engineering*, 1961.

Kantorovitch, L. On the translocation of masses. *Management Science*, 1958.

Kingma, D. and Ba, J. Adam: A method for stochastic optimization. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2015.

Larkin, M. A., Blackshields, G., Brown, N. P., Chenna, R., McGettigan, P. A., McWilliam, H., Valentin, F., Wallace, I. M., Wilm, A., Lopez, R., et al. Clustal W and Clustal X version 2.0. *Bioinformatics*, 2007.

Levenshtein, V. I. Binary codes capable of correcting deletions, insertions, and reversals. In *Doklady Akademii Nauk*, 1965.

Levina, E. and Bickel, P. The Earth Mover's distance is the Mallows distance: Some insights from statistics. In *Proceedings of the Eighth IEEE International Conference on Computer Vision (ICCV)*, 2001.

Lewis, J. *A Dynamic Load Balancing Approach to the Control of Multiserver Polling Systems with Applications to Elevator System Dispatching*. PhD thesis, University of Massachusetts, Amherst, 1991.

Lewis, P. A. and Shedler, G. S. Simulation of nonhomogeneous Poisson processes by thinning. *Naval Research Logistics Quarterly*, 1979.

Lin, C.-C. and Eisner, J. Neural particle smoothing for sampling from conditional sequence models. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)*, 2018.

Linderman, S. W., Wang, Y., and Blei, D. M. Bayesian inference for latent Hawkes processes. In *Advances in Approximate Bayesian Inference Workshop, 31st Conference on Neural Information Processing Systems*, 2017.

Liniger, T. J. *Multivariate Hawkes processes*. Diss., Eidgenössische Technische Hochschule ETH Zürich, Nr. 18403, 2009.

Listgarten, J., Neal, R. M., Roweis, S. T., and Emili, A. Multiple alignment of continuous time series. In *Advances in Neural Information Processing Systems (NIPS)*, 2005.

Little, R. J. A. and Rubin, D. B. *Statistical Analysis with Missing Data*. 1987.

Liu, J. S. and Chen, R. Sequential Monte Carlo methods for dynamic systems. *Journal of the American Statistical Association*, 1998.

McLachlan, G. and Krishnan, T. *The EM algorithm and Extensions*. 2007.

Mei, H. and Eisner, J. The neural Hawkes process: A neurally self-modulating multivariate point process. In *Advances in Neural Information Processing Systems (NIPS)*, 2017.

Mohan, K. and Pearl, J. Graphical models for processing missing data. *arXiv preprint arXiv:1801.03583*, 2018.

Moral, P. D. Nonlinear filtering: Interacting particle resolution. *Comptes Rendus de l'Academie des Sciences-Serie I-Mathematique*, 1997.

Mount, D. W. *Bioinformatics: Sequence and Genome Analysis*. 2004.

Notredame, C., Higgins, D. G., and Heringa, J. T-Coffee: A novel method for fast and accurate multiple sequence alignment. *Journal of molecular biology*, 2000.

Peters, M., Neumann, M., Iyyer, M., Gardner, M., Clark, C., Lee, K., and Zettlemoyer, L. Deep contextualized word representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, 2018.

Rabiner, L. R. A tutorial on hidden Markov models and selected applications in speech recognition. In *Proceedings of the IEEE*, 1989.

Rao, V. and Teh, Y. W. MCMC for continuous-time discrete-state systems. In *Advances in Neural Information Processing Systems*, 2012.

Rao, V. and Teh, Y. W. Fast MCMC sampling for Markov jump processes and extensions. *The Journal of Machine Learning Research*, 2013.

Rauch, H. E., Striebel, C. T., and Tung, F. Maximum likelihood estimates of linear dynamic systems. *AIAA Journal*, 1965.

Sakoe, H. and Chiba, S. A dynamic programming approach to continuous speech recognition. In *Proceedings of the Seventh International Congress on Acoustics, Budapest*, 1971.

Shelton, C. R., Qin, Z., and Shetty, C. Hawkes process inference with missing data. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2018.

Spall, J. C. *Introduction to Stochastic Search and Optimization: Estimation, Simulation, and Control*. 2005.

Villani, C. *Optimal Transport: Old and New*. 2008.

Wei, G. C. G. and Tanner, M. A. A Monte Carlo implementation of the EM algorithm and the poor man's data augmentation algorithms. *Journal of the American Statistical Association*, 1990.

Whong, C. FOILing NYC's taxi trip data, 2014.

Williams, R. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 1992.

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

Appendices

A. Little & Rubin (1987)'s Missing-Data Taxonomy

Little & Rubin (1987)'s classical taxonomy of MNAR, MAR, and MCAR mechanisms[12] was meant for graphical models. A graphical model has a *fixed set* of random variables. The missingness mechanisms envisioned by Little & Rubin (1987) simply decide which of those variables are suppressed in a joint observation. For them, an observed sample always reveals *which* variables were observed, and thus it reveals *how many* variables are missing.

In contrast, our incomplete event stream is most simply described as a single random variable Y that is *partly* missing. If we tried to describe it using $|\mathbf{x} \sqcup \mathbf{z}|$ random variables with values like $k@t$, then the observed sample $\mathbf{x}$ would *not* reveal the number of missing variables $|\mathbf{z}|$ nor the total number of variables $|\mathbf{x} \sqcup \mathbf{z}|$. There would not be a fixed set of random variables.

To formulate our model in Little & Rubin's terms, we would need a fixed set of uncountably many random variables K_t where t ranges over the set of times. $K_t = k$ if there is an event of type k at time t, and otherwise $K_t = 0$. For some finite set of times t, we observe a specific value $K_t > 0$, corresponding to some observed event. For all other times t, the value of K_t is missing, meaning that we do not know whether or not there is any event at time t, let alone the type of such an event. A crucial point is that 0 values are never observed in our setting, because we are never told that an event did *not* happen at time t. In contrast, a value > 0 (corresponding to an event) may be either observed or unobserved. Thus, the probability that K_t is missing depends on whether $K_t > 0$, meaning that this setting is MNAR.

We preferred to present our model (section 2.1) in terms of the finite sequences that are generated or read by our LSTMs. This simplified the notation later in the paper. But it does not cure the MNAR property: see section 5.1.

Again, our presentation does not allow a Little & Rubin (1987) style formulation in terms of a finite fixed set of random variables, some of which have missing values. That formulation would work if we knew the total number of events I, and were simply missing the value k_i and/or t_i for some indices i. But in our setting, the number of events is itself missing: after each observed event i, we are missing J_i events where J_i is itself unknown. In other words, we need to impute even the number of variables in the complete sequence $\mathbf{x} \sqcup \mathbf{z}$, not just their values.

Our definition of MAR in section 2.1 is the correct generalization of Little & Rubin (1987)'s definition: namely, it is the case in which the second factor of equation (1) can be ignored. The ability to ignore that factor is precisely why anyone cares about the MAR case. This was mentioned at equation (1), and is discussed in conjunction with the EM algorithm in Appendix H.

Since missing-event settings tend to violate this desirable MAR property, all our experiments address MNAR problems. As Little & Rubin (1987) explained, the more general case of MNAR data cannot be treated without additional knowledge. The difficulty is that identifying p_{model} *jointly* with p_{miss} becomes impossible. If you observe few 50-year-olds on your survey, you cannot know (beyond your prior) whether that's because there are few 50-year-olds, or because 50-year-olds are very likely to omit their age.

Fortunately, we do have additional knowledge in our setting. Joint identification of p_{model} and p_{miss} is unnecessary if either (1) one has separate knowledge of the missingness distribution p_{miss}, or (2) one has separate knowledge of the complete-data distribution p_{model}. In fact, both (1) and (2) hold in the MNAR experiments of this paper (sections 5.1–5.2). But in general, if we know at least one of the distributions, then we can still infer the other (Appendix H).

A.1. Obtaining Complete Data

Readers might wonder why (2) above would hold in a missing-data situation. In practice, where would we obtain a dataset of complete event streams (as in section 5.2) for supervised training of $p_{\text{model}}(\mathbf{x} \sqcup \mathbf{z})$?

In some event stream scenarios, a training dataset of complete event streams can be collected at extra expense. This is the hope in the medical and user-interface scenarios in section 1. For our imputation method to work on partially observed streams $\mathbf{x} \sqcup \mathbf{z}$, their complete streams should be distributed like the ones in the training dataset.

Other scenarios could be described as having **eventually complete** streams. Complete information about each event stream *eventually* arrives, at no extra expense, and that event stream can then be used for training. For example, in the competitive game scenario in section 1), perhaps including wars and political campaigns, each game's true complete history is revealed after the game is over and the need for secrecy has passed. While a game is underway, however, some events are still missing, and imputing them is valuable. Both (1) and (2) hold in these settings.

[12] **Missing not at random (MNAR)** makes no assumptions about the missingness mechanism. **Missing at random (MAR)** is a modeling assumption: determining from data whether the MAR property holds is "almost impossible" (Mohan & Pearl, 2018). **Missing completely at random (MCAR)** is a simple special case of MAR.

An interesting subclass of eventual completeness arises in monitoring settings such as epidemiology, journalism, and sensor networks. These often have **reporting delays**, so that one observes each event only some time after it happens. Yet one must make decisions at time $t < T$ based on the events that have been observed so far. This may involve imputing the past and predicting the future. The missingness mechanism for these reporting delays says that more recent events (soon before the current time t) are more likely to be missing. The probability that such an event would be missing depends on the specific distribution of delays, which can be learned with supervision once all the data have arrved.

We point out that in all these cases, the "complete" streams $\mathbf{x} \sqcup \mathbf{z}$ that are used to train p_{model} do not actually have to be *causally* complete. It may be that in the real world, there are additional latent events $\mathbf{w}$ that cause the events in $\mathbf{x} \sqcup \mathbf{z}$ or mediate their interactions. Mei & Eisner (2017, section 6.3) found that the neural Hawkes process was expressive enough in practice to ignore this causal structure and simply use $\mathbf{x} \sqcup \mathbf{z}$ streams to directly train a neural Hawkes process model $p_{\text{model}}(\mathbf{x} \sqcup \mathbf{z})$ of the *marginal* distribution of $\mathbf{x} \sqcup \mathbf{z}$, without explicitly considering $\mathbf{w}$ in the model or attempting to sum over $\mathbf{w}$ values. The assumption here is the usual assumption that $\mathbf{x} \sqcup \mathbf{z}$ will have the same distribution in training and test data, and thus $\mathbf{w}$ will be missing in both, with the same missingness mechanism in both. By contrast, $\mathbf{z}$ is missing only in test data. It is not possible to impute $\mathbf{w}$ because it was not modeled explicitly, nor observed even in training data. However, it remains possible to impute $\mathbf{z}$ in test data based on its distribution in training data.

B. Complete Data Model Details

Our complete data model, such as a neural Hawkes process, gives the probability $p_{\text{model}}(\mathbf{x} \sqcup \mathbf{z})$ that $\mathbf{x} \sqcup \mathbf{z}$ will be the complete set of events on a given interval $[0, T)$. this probability can always be written in the factored form

$$\Big(\prod_{i=0}^{I}\prod_{j=0}^{J_i} p(k_{i,j} @ t_{i,j} \mid \mathcal{H}(t_{i,j}))\Big) \cdot p(@ \geq T \mid \mathcal{H}(T)) \quad (14)$$

where $p(k @ t \mid \mathcal{H}(t))$ denotes the probability density that the first event following $\mathcal{H}(t)$ (which is the set of events occurring *strictly* before t) will be $k @ t$, and $p(@ \geq t' \mid \mathcal{H}(t))$ denotes the probability that this event will fall at some time $\geq t'$.

Thus, the final factor of (14) is the probability that there are no more events on $[0, T)$ following the last event of $\mathbf{x} \sqcup \mathbf{z}$. The initial factor $p(k_0 @ t_0 \mid \mathcal{H}(t_0))$ is defined to be 1, since the boundary event $k_0 @ t_0$ is given (see section 2.1).

B.1. Neural Hawkes Process Details

In this section we elaborate slightly on section 2.2. Again, $\lambda_k(t \mid \mathcal{H}(t))$ is defined by equation (2) in terms of the hidden state of a *continuous-time* left-to-right LSTM. We spell out the continuous-time LSTM equations here; more details about them may be found in Mei & Eisner (2017).

$$\mathbf{h}(t) = \mathbf{o}_i \odot (2\sigma(2\mathbf{c}(t)) - 1) \text{ for } t \in (t_{i-1}, t_i] \quad (15)$$

where the interval $(t_{i-1}, t_i]$ has consecutive observations $k_{i-1} @ t_{i-1}$ and $k_i @ t_i$ as endpoints. At t_i, the continuous-time LSTM reads $k_i @ t_i$ and updates the current (decayed) hidden cells $\mathbf{c}(t)$ to new initial values $\mathbf{c}_{i+1}$, based on the current (decayed) hidden state $\mathbf{h}(t_i)$, as follows:[13]

$$\mathbf{i}_{i+1} \leftarrow \sigma\left(\mathbf{W}_{\mathrm{i}}\mathbf{k}_i + \mathbf{U}_{\mathrm{i}}\mathbf{h}(t_i) + \mathbf{d}_{\mathrm{i}}\right) \quad (16a)$$
$$\mathbf{f}_{i+1} \leftarrow \sigma\left(\mathbf{W}_{\mathrm{f}}\mathbf{k}_i + \mathbf{U}_{\mathrm{f}}\mathbf{h}(t_i) + \mathbf{d}_{\mathrm{f}}\right) \quad (16b)$$
$$\mathbf{z}_{i+1} \leftarrow 2\sigma\left(\mathbf{W}_{\mathrm{z}}\mathbf{k}_i + \mathbf{U}_{\mathrm{z}}\mathbf{h}(t_i) + \mathbf{d}_{\mathrm{z}}\right) - 1 \quad (16c)$$
$$\mathbf{o}_{i+1} \leftarrow \sigma\left(\mathbf{W}_{\mathrm{o}}\mathbf{k}_i + \mathbf{U}_{\mathrm{o}}\mathbf{h}(t_i) + \mathbf{d}_{\mathrm{o}}\right) \quad (16d)$$

$$\mathbf{c}_{i+1} \leftarrow \mathbf{f}_{i+1} \odot \mathbf{c}(t_i) + \mathbf{i}_{i+1} \odot \mathbf{z}_{i+1} \quad (17a)$$
$$\underline{\mathbf{c}}_{i+1} \leftarrow \underline{\mathbf{f}}_{i+1} \odot \underline{\mathbf{c}}_i + \underline{\mathbf{i}}_{i+1} \odot \mathbf{z}_{i+1} \quad (17b)$$
$$\boldsymbol{\delta}_{i+1} \leftarrow f\left(\mathbf{W}_{\mathrm{d}}\mathbf{k}_i + \mathbf{U}_{\mathrm{d}}\mathbf{h}(t_i) + \mathbf{d}_{\mathrm{d}}\right) \quad (17c)$$

The vector $\mathbf{k}_i \in \{0, 1\}^K$ is the i^{th} input: a one-hot encoding of the new event k_i, with non-zero value only at the entry indexed by k_i. Then, $\mathbf{c}(t)$ for $t \in (t_{i-1}, t_i]$ is given by (18), which continues to control $\mathbf{h}(t)$ except that i has now increased by 1).

$$\mathbf{c}(t) \stackrel{\text{def}}{=} \underline{\mathbf{c}}_{i+1} + \left(\mathbf{c}_{i+1} - \underline{\mathbf{c}}_{i+1}\right) \exp\left(-\boldsymbol{\delta}_{i+1}\left(t - t_i\right)\right) \quad (18)$$

On the interval $(t_i, t_{i+1}]$, $\mathbf{c}(t)$ follows an exponential curve that begins at $\mathbf{c}_{i+1}$ (in the sense that $\lim_{t \to t_i^+} \mathbf{c}(t) = \mathbf{c}_{i+1}$) and decays, as time t increases, toward $\underline{\mathbf{c}}_{i+1}$ (which it would approach as $t \to \infty$, if extrapolated).

The **intensity** $\lambda_k(t \mid \mathcal{H}(t)) \in \mathbb{R}_{\geq 0}$ may be thought of as the instantaneous *rate* of events of type k at time t. More precisely, as $dt \to 0^+$, the expected number of events of type k occurring in the interval $[t, t + dt)$, divided by dt, approaches $\lambda_k(t \mid \mathcal{H}(t))$. If no event of any type occurs in this interval (which becomes almost sure as $dt \to 0^+$), one may still occur in the next interval $[t + dt, t + 2dt)$, and so on. The intensity functions $\lambda_k(t \mid \mathcal{H}(t))$ are continuous on intervals during which no event occurs (note that $\mathcal{H}(t)$ is constant on such intervals). They jointly determine a distribution over the time of the next event after $\mathcal{H}(t)$, as used in every factor of equation (14). As it turns out (Mei

[13] The upright-font subscripts i, f, z and o are not variables, but constant labels that distinguish different $\mathbf{W}$, $\mathbf{U}$ and $\mathbf{d}$ tensors. The $\underline{\mathbf{f}}$ and $\underline{\mathbf{i}}$ in equation (17b) are defined analogously to $\mathbf{f}$ and $\mathbf{i}$ but with different weights.

& Eisner, 2017), $\log p_{\text{model}}(Y = \mathbf{x} \sqcup \mathbf{z})$ becomes

$$\sum_{\ell} \log \lambda_{k_\ell}(t_\ell \mid \mathcal{H}(t_\ell)) - \int_{t=0}^{T} \sum_{k=1}^{K} \lambda_k(t \mid \mathcal{H}(t))dt \quad (19)$$

where the first sum ranges over all events $k_\ell @ t_\ell$ in $\mathbf{x} \sqcup \mathbf{z}$.

We can therefore train the parameters $\boldsymbol{\theta}$ of the λ_k functions by maximizing log-likelihood on training data. The first term of equation (19) can be differentiated by backpropagation. Mei & Eisner (2017) explain how simple Monte Carlo integration (see also our Appendix C.3) gives an unbiased estimate of the second term of equation (19), and how the random terms in the Monte Carlo estimate can similarly be differentiated to give a stochastic gradient.

C. Sequential Monte Carlo Details

Our main algorithm is presented as Algorithm 1. It covers both particle filtering and particle smoothing, with optional multinomial resampling.

In this section, we provide some additional details and notes on the design and operation of the pseudocode.

C.1. Explicit Formula for the Proposal Distribution

The proposal distribution $q(\mathbf{z} \mid \mathbf{x})$ factors as follows, and the pseudocode uses this factorization to construct $\mathbf{z}$ by sampling its individual events from left to right:

$$\prod_{i=0}^{I} \Big(\prod_{j=1}^{J_i} \big(q(k_{i,j} @ t_{i,j} \mid \mathcal{H}(t_{i,j}), \mathcal{F}(t_{i,j}))\big) \quad (20)$$
$$\cdot\, q(@ \geq t_{i+1} \mid \mathcal{H}(t_{i+1}), \mathcal{F}(t_{i,J_i}))\Big)$$

Here the notation for $q(\cdot \mid \cdot)$ is the same as that for $p(\cdot \mid \cdot)$ in Appendix B. However, the $q(\cdot \mid \cdot)$ terms are proposal probabilities that condition on different evidence—not only the set $\mathcal{H}(t)$ of all events (observed and unobserved) at times $< t$, but also the set $\mathcal{F}(t)$ of events at times $> t$.[14] All of the proposal probabilities $q(\cdot \mid \cdot)$ are determined by the intensity functions in (5).

We can sample $\mathbf{z}$ from $q(\mathbf{z} \mid \mathbf{x})$ in chronological order: for each $0 \leq i \leq I$ in turn, draw a sequence of J_i unobserved events that follow the observed event $k_i @ t_i$. The probabilities of these J_i events are the inner factors in equation (20). This sequence ends (thereby determining J_i) if the next proposed event would have fallen after t_{i+1} and thus is preempted by the observed event $k_{i+1} @ t_{i+1}$. The probability of so ending the sequence corresponds to the $q(@ \geq t_{i+1} \mid \cdots)$ factor in equation (20).

[14] In particular, the second q factor above is the probability that the event at time t_{i,J_i} is the last one before t_{i+1}, given knowledge of all past events up through and including the one at t_{i,J_i}, and all future observed events starting with the one at t_{i+1}.

Equation (20) resembles equation (14), but it conditions each proposed unobserved event not only on the history but also on the future. Section 3.2.1 tries to train $q(\mathbf{z} \mid \mathbf{x})$ to approximate the target distribution $p(\mathbf{z} \mid \mathbf{x})$, by making $q(\cdot \mid \mathcal{H}, \mathcal{F}) \approx p(\cdot \mid \mathcal{H}, \mathcal{F})$. In other words, at each step, q should draw the next proposed event approximately from the posterior of the model p, even though we have no closed form computation for that posterior.

Just as equation (14) yields the formula (19) for $\log p_{\text{model}}$ when we use a neural Hawkes process model, equation (20) yields the following formula for $\log q(\mathbf{z} \mid \mathbf{x})$ when we use the proposal intensities from (5):

$$\sum_{\ell} \log \lambda^q_{k_\ell}(t_\ell \mid \mathcal{H}(t_\ell), \mathcal{F}(t_\ell))$$
$$- \int_{t=0}^{T} \sum_{k=1}^{K} \lambda^q_k(t \mid \mathcal{H}(t), \mathcal{F}(t))\, dt \quad (21)$$

where the first sum ranges over all events $k_\ell @ t_\ell$ in $\mathbf{z}$ only.

C.2. Managing LSTM State Information

The push and pop operations shown in the pseudocode must be implemented so that they also have the effect of updating LSTM configurations.

Our p_{model} uses a left-to-right LSTM to construct its state after reading all events so far from left to right (section 2.2). Since each particle posits a different event sequence, we maintain a separate LSTM configuration for each particle $m = 1, 2, \ldots, M$. If *smooth* = **true**, our q additionally uses a right-to-left LSTM whose state has read all future observed events from right to left (section 3.2). We maintain the configuration of this LSTM as well.

Specifically, in Algorithm 1, when we push an event to the stack $\mathcal{H}_m$ (lines 5 and 46), we update the configuration of particle m's left-to-right LSTM (including gates, cell memories and states).

If *smooth* = **true**, then when we push an event to the stack $\mathcal{F}$ (line 8), we update the configuration of the right-to-left LSTM. Moreover, before updating that configuration, we push it onto an parallel stack, so that we can revert the update when we later *pop* the event from $\mathcal{F}$ (line 36).

These LSTM configurations provide the $\mathbf{h}(t)$ and $\bar{\mathbf{h}}(t)$ vectors for the computation of intensities $\lambda^p_k(t)$ and $\lambda^q_k(t)$ in equations (2) and (5). These intensities are needed in lines 43 and 45 of the algorithm.

C.3. Integral Computation

Mei & Eisner (2017, section B.2) construct a Monte Carlo estimator of the $\int_0^T$ integral in equation (19), by evaluating $\sum_k \lambda_k(t \mid \mathcal{H}(t)) \cdot T$ at a random $t \sim \text{Unif}(0, 1)$. While even one such sample would provide an unbiased estimate,

Algorithm 1 Sequential Monte Carlo — Neural Hawkes Particle Filtering/Smoothing

Input: observed sequence $\mathbf{x} = k_0@t_0, \ldots, k_{I+1}@t_{I+1}$ with $t_0 = 0, t_{I+1} = T$;
model p; missingness mechanism p_{miss}; proposal distribution q; number of particles M;
boolean flags *smooth* and *resample*

Output: collection $\{(\mathbf{z}_1, w_1), \ldots, (\mathbf{z}_M, w_M)\}$ of weighted particles

procedure SEQUENTIALMONTECARLO($\mathbf{x}, p, p_{\text{miss}}, q, M, smooth, resample$)
  for $m = 1$ **to** M : ▷ *initialize the M weighted particles $(\mathbf{z}_m, w_m)$*
    $\mathbf{z}_m \leftarrow$ empty seq; $w_m \leftarrow 1$
    ▷ *history $\mathcal{H}_m$ will be a stack of the past events, namely $k_0@t_0$ followed by the prefix of $\mathbf{x} \sqcup \mathbf{z}_m$ generated so far*
    $\mathcal{H}_m \leftarrow$ empty stack; push $k_0@t_0$ onto $\mathcal{H}_m$ ▷ *boundary event 0 (not generated by p_{model})*
  $\mathcal{F} \leftarrow$ empty stack ▷ *$\mathcal{F}$ is a stack of the future observed events, with the next event on top*
  for $i = I$ **downto** 0 : ▷ *initialize $\mathcal{F}$; later, as we reach each event, we'll pop it from $\mathcal{F}$ and push it onto $\mathcal{H}_m$ ($\forall m$)*
    push $k_{i+1}@t_{i+1}$ onto $\mathcal{F}$
  for $i = 0$ **to** I : ▷ *propose unobserved events on interval (t_i, t_{i+1}), then observe next event $k_{i+1}@t_{i+1}$*
    for $m = 1$ **to** M :
      DRAWSEGMENT(i, m) ▷ *destructively extend $\mathbf{z}_m, w_m, \mathcal{H}_m$ with events on $(t_i, t_{i+1}]$*
    if *resample* & LOWESS() : RESAMPLE() ▷ *optional multinomial resampling replaces all weighted particles*
  return $\{(\mathbf{z}_m, w_m / \sum_{m=1}^M w_m)\}_{m=1}^M$ ▷ *M particles with weights normalized as in equation (3)*
procedure LOWESS ▷ *check if effective sample size is low*
  ESS $\leftarrow (\sum_{m=1}^M w_m)^2 / \sum_{m=1}^M (w_m)^2$
  if ESS $< M/2$: **return true**
  return false
procedure RESAMPLE ▷ *has access to global variables*
  for $m = 1$ **to** M : ▷ *often draws multiple copies of good (high-weight) particles, 0 copies of bad ones*
    draw $\tilde{m} \in \{1, \ldots, M\}$ where probability of choosing any $\tilde{m}$ is proportional to $w_{\tilde{m}}$; then set $\tilde{\mathbf{z}}_m \leftarrow \mathbf{z}_{\tilde{m}}$
  for $m = 1$ **to** M :
    $\mathbf{z}_m \leftarrow \tilde{\mathbf{z}}_m$; $w_m \leftarrow 1$ ▷ *update particles and their weights*
procedure DRAWSEGMENT(i, m) ▷ *has access to global variables*
  ▷ *algorithm input p gives info to define intensity function $\lambda_k^p(t) \stackrel{\text{def}}{=} \lambda_k(t \mid \mathcal{H}_m)$*
  ▷ *algorithm input q gives info to define intensity function $\lambda_k^q(t) \stackrel{\text{def}}{=} \lambda_k(t \mid \mathcal{H}_m, \mathcal{F})$, or simply $\lambda_k^q(t) = \lambda_k^p(t)$ if $smooth =$ **false***
  ▷ *these functions consult the state of a left-to-right LSTM that's read $\mathcal{H}_m$ and possibly a right-to-left LSTM that's read $\mathcal{F}$*
  ▷ *we also define the **total intensity functions** $\lambda^p(t) \stackrel{\text{def}}{=} \sum_{k=1}^K \lambda_k^p(t)$ and $\lambda^q(t) \stackrel{\text{def}}{=} \sum_{k=1}^K \lambda_k^q(t)$*
  $i' \leftarrow i$; $j \leftarrow 0$; $t \leftarrow t_i$ ▷ *t_i can be found as the time of the top element of $\mathcal{H}_m$ (currently an observed event)*
  repeat ▷ *each iteration adds a new event to $\mathcal{H}_m$ with index $\langle i', j \rangle = \langle i, 1 \rangle, \ldots, \langle i, J_i \rangle, \langle i+1, 0 \rangle$*
    $j \leftarrow j + 1$
    repeat ▷ *thinning algorithm (see Mei & Eisner, 2017)*
      find any $\lambda^* \geq \sup \{\lambda^q(t') : t' \in (t, t_{i+1}]\}$ ▷ *e.g., old λ^* still works if i unchanged; see Appendix C.4*
      draw $\Delta \sim \text{Exp}(\lambda^*)$, $u \sim \text{Unif}(0, 1)$
      $t \mathrel{+}= \Delta$ ▷ *time of next proposed event (before thinning)*
      if $t > t_{i+1}$: ▷ *t_{i+1} can be found as the time of the top element of $\mathcal{F}$ (always an observed event)*
        $k@t \leftarrow$ pop $\mathcal{F}$; $i' \leftarrow i + 1$; $j \leftarrow 0$; ▷ *preempt proposed event by $k_{i+1}@t_{i+1}$ (popped from future into present)*
        break
    until $u\lambda^* \leq \lambda^q(t)$ ▷ *thinning: accept proposed time t only with prob $\frac{\lambda^q(t)}{\lambda^*} \leq 1$*
    ▷ *we've now chosen next event time $t_{i',j}$ to be t; let t_{prev} denote the time of the top element of $\mathcal{H}_m$*
    if $i' = i$: ▷ *it's a missing event*
      draw $k \in \{1, \ldots, K\}$ where probability of k is proportional to $\lambda_k^q(t)$ ▷ *choose event type for the proposed time*
      append $k@t$ to $\mathbf{z}_m$ ▷ *add our proposed event $k_{i',j}@t_{i',j}$*
      $w_m \leftarrow w_m \,/\, (\exp(-\int_{t'=t_{\text{prev}}}^t \lambda^q(t')\, dt') \cdot \lambda_k^q(t))$ ▷ *new factor within q in denominator of (3); see Appendix C.3*
    if $i' \leq I$: ▷ *skip final boundary event $I+1$ (not generated by p_{model})*
      $w_m \leftarrow w_m \cdot (\exp(-\int_{t'=t_{\text{prev}}}^t \lambda^p(t')\, dt') \cdot \lambda_k^p(t))$ ▷ *new factor within p_{model} in numerator of (3); see Appendix C.3*
      push $k@t$ onto $\mathcal{H}_m$ ▷ *event $\langle i, j \rangle$ just generated now becomes part of the past*
      $w_m \leftarrow w_m \cdot p_{\text{miss}}((k@t \in Z) = (i' = i) \mid \mathcal{H}_m)$ ▷ *new factor within p_{miss} in numerator of (3): missing or obs*
  until $i' = i + 1$

they draw $N = O(I)$ such samples, where I is the number of events, and average over these samples. This reduces the variance of the estimator, which decreases as $O(1/N)$. Notice that because they sample the N time points uniformly on $[0, T)$, longer intervals between observed events will tend to contain more points, which is appropriate.

Mei & Eisner (2017) (Appendix C.2) found that rather few samples could be used to estimate the integral. Indeed, even sampling at only I time points gave a standard deviation of log-likelihood—for the whole sequence—that was on the order of 0.1% of absolute (Mei, p.c.).

Our particle methods in the present paper involve *comparing probabilities*. For each observed sequence $\mathbf{x}$, we use (3) to reweight the M particles according to their probability under the model divided by their probability under the proposal distribution. This means contrasting *two* probabilities for each particle (the p and q probabilities). It also means *comparing* the resulting probability ratios across all M particles, resulting in the normalized weights of equation (3).

For each of the M particles, the p_{model} factor in equation (3) is obtained by exponentiating equation (19), while the q factor is obtained by exponentiating equation (21). This means that each of these $2M$ factors contains the exp of an integral. To make all of these integral estimates more comparable and thus reduce the variance in the importance weights w_m (equation (3)), we evaluate all $2M$ integrals at the same set of N time points (see Appendix G.8). This practice ensures a "paired comparison" among particles: w_m and $w_{m'}$ differ only because they have different intensities at the sampled points, and not also because they sample at different points.

In Algorithm 1, these integral estimates are accumulated gradually at lines 43 and 45. The idea is that particle $\mathbf{z}_m$ partitions $[0, T)$ into the intervals between successive events of $\mathbf{x} \sqcup \mathbf{z}_m$. Thus, the (estimated) integral over $[0, T)$ can be expressed by summing the (estimated) integrals over these intervals. The estimate over an interval uses only the small subset of the N time points that fall into the interval. When we exponentiate the integrals to convert from log-probabilities into probabilities, this sum turns into a product, as shown at lines 43 and 45.

This gradual accumulation method gives the same result as if we had computed each integral "all at once" before exponentiating. However, it is useful to begin weighting the particles before they are complete. After each event $k_i @ t_i$ (for $0 \le i \le I + 1$), the partial particles up through this event already have partial weights w_m. It is these partial weights that are used by the RESAMPLE procedure (when $resample =$ **true**).

In all experiments in this paper, we first sampled $I + 1$ points uniformly on $[0, T)$, for an average of only 1 time point per interval. In addition, for each interval (t_i, t_{i+1}), we sampled 1 point uniformly on that interval if it did not yet contain any points. Thus, $N \in [I + 1, 2I + 1]$.

Sampling at more points might be wise in settings where there are many missing events per interval (e.g., large ρ in section 5.1). This is especially true when $resample =$ **true**. Resampling allows us to try multiple extensions of a high-weight particle; at the next resampling step, we prefer to keep the extensions that fared best. The danger is that if only a few sampling points happen to fall between resampling steps, then we may make a poor (high-variance) estimate of which extensions fared best.

For our setting, however, we found only negligible changes in the results by increasing to 5 time points per interval (i.e., sampling $5I+5$ points at the first step). Our evaluation metric (the minimum of (13) over all alignments $\mathbf{a}$) became slightly better for some values of C and slightly worse for others, but never by more than 2% relative. This is about the same variance that we get across different runs (with different random seeds) that have 1 time point per interval.

Thus, we report only the results of the faster scheme. We caution that other settings might be more sensitive to this hyperparameter settings. Thus, it might be wise to eliminate the hyperparameter altogether by estimating the integrals at lines 43 and 45 with a more sophisticated Monte Carlo integration method, such as the adaptive partitioning method of Baran et al. (2008), which can bound the additive error of the estimate with high probability. This approach no longer provides a "paired comparison" across particles, nor does it need one.

C.4. Choice of λ^*

How do we construct the upper bound λ^* (line 32 of Algorithm 1)? For particle filtering, we follow the recipe in Appendix B.3 of Mei & Eisner (2017): we can express $\lambda^* = f_k(\max_t g_1(t) + \ldots + \max_t g_n(t))$ where each summand $v_{kd} h_d(t) = v_{kd} \cdot o_{id} \cdot (2\sigma(2c_d(t)) - 1)$ is upper-bounded by $\max_{c \in \{c_{id}, \bar{c}_{id}\}} v_{kd} \cdot o_{id} \cdot (2\sigma(2c) - 1)$. Note that the coefficients v_{kd} may be either positive or negative.

For particle smoothing, we simply have more summands inside f_k so $\lambda^* = f_k(\max_t g_1(t) + \ldots + \max_t g_n(t) + \max_t \bar{g}_1(t) + \ldots + \max_t \bar{g}_{\bar{n}}(t))$ where each extra summand $u_{kd} \bar{h}_d(t) = u_{kd} \cdot \bar{o}_{id} \cdot (2\sigma(2\bar{c}_d(t)) - 1)$ is upper-bounded by $\max_{c \in \{\bar{c}_{id}, \bar{\bar{c}}_{id}\}} u_{kd} \cdot \bar{o}_{id} \cdot (2\sigma(2c) - 1)$ and each u_{kd} is the d^{th} element of vector $\mathbf{v}_k^\top \mathbf{B}$ (equation (5)). Note that the $\bar{o}_{id}, \bar{c}_{id}, \bar{\bar{c}}_{id}$ of newly added summands $\bar{g}$ are actually from the right-to-left LSTM while those of g are from the left-to-right LSTM.

C.5. Missing Data Factors in p

Recall that the joint model (1) includes a factor $p_{\text{miss}}(\mathbf{z} \mid \mathbf{x} \sqcup \mathbf{z})$, which appears in the numerator of the unnormalized importance weight (3). Regardless of the form of this factor, it could be multiplied into the particle's weight $\tilde{w}_m$ at the *end* of sampling (line 13 of Algorithm 1).

However, for some p_{miss} distributions, there is a better way. Algorithm 1 assumes that the missingness of each event $k@t$ depends only on that event and preceding events,[15] so that $p_{\text{miss}}(\mathbf{z} \mid \mathbf{x} \sqcup \mathbf{z})$ factors as

$$\prod_{\ell \in \text{indices}(\mathbf{z})} p_{\text{miss}}(k_\ell @ t_\ell \in Z \mid \{k_{\ell'} @ t_{\ell'} : \ell' \leq \ell\}) \cdot \prod_{\ell \in \text{indices}(\mathbf{x})} p_{\text{miss}}(k_\ell @ t_\ell \notin Z \mid \{k_{\ell'} @ t_{\ell'} : \ell' \leq \ell\}) \tag{22}$$

Algorithm 1 can thus *incrementally* incorporate the subfactors of equation (22), and does so at line 47 of Algorithm 1. For example, with the missingness mechanism in our experiments, equation (12), the p_{miss} factor in line 47 is ρ_k if the event is unobserved (that is, $i' = i$) or $1 - \rho_k$ if it is observed.

These subfactors are therefore taken into account as the particles are constructed, and thus play a role in resampling.

C.6. Optional Missing Data Factors in q

We can optionally improve the particle filtering proposal intensities to incorporate the p_{miss} factor discussed in Appendix C.5 (in which case that factor will be multiplied into the denominator of (3) and not just the numerator). This makes $q(\mathbf{z} \mid \mathbf{x})$ better match $p(\mathbf{z} \mid \mathbf{x})$: it means we will rarely posit an unobserved event that would rarely have gone missing.

Specifically, if a completed-data event $k@t$ would have probability $\rho_k(t \mid \mathcal{H}(t))$ of going missing given the preceding events $\mathcal{H}(t)$, it is wise to define $\lambda_k^q(t \mid \mathcal{H}(t)) = \lambda_k^p(t \mid \mathcal{H}(t)) \cdot \rho_k(t \mid \mathcal{H}(t))$.

We do include this extra ρ_k factor when defining λ_k^q for our experiments (section 5); that is, we modify the definition of λ_k^q at line 25. The factor is particularly simple in our experiments, where ρ_k is constant for each k.

Was this factor really necessary in the case of particle smoothing? One might say no: particle smoothing already tries to ensure through training that the proposal distribution will incorporate p_{miss}. That is because section 3.2.1 aims to train $\lambda_k^q(t \mid \mathcal{H}(t), \mathcal{F}(t))$ so that the resulting $q(\mathbf{z} \mid \mathbf{x}) \approx p(\mathbf{z} \mid \mathbf{x})$, and the posterior distribution $p(\mathbf{z} \mid \mathbf{x})$ does condition on the missingness of $\mathbf{z}$.

Still, if the ρ_k factor is known, why not include it explicitly in the proposal distribution, instead of having to train the BiLSTM to mimic it? Thus, in effect, we have modified the right-hand side of equation (5) to include a factor of ρ_k. This yields a more expressive and better-factored family of proposal distributions: missingness is now handled by the known ρ_k factor and the BiLSTM does not have to explain it. Additionally, our proposal distribution becomes more conservative about proposing missing events, because having a lot of missing events is *a posteriori* improbable. In other words, p_{miss} as given in equation (12) falls off with the number of missing events $|\mathbf{z}|$.

Modifying equation (5) in this way is particularly useful in the special case $\rho_k = 0$ (i.e., event type k is never missing and should not be proposed). There, it enforces the hard constraint that $\lambda_k^q = 0$ (something that the BiLSTM by itself could not achieve); and since this constraint is enforced regardless of the BiLSTM parameters, the events of type k appropriately become irrelevant to the training of the BiLSTM, which can focus on predicting other event types.

C.7. Events with Equal Times

In contrast to the notation in the main paper, our pseudocode is written in terms of sequence of events, rather than sets of events. As a result, it can handle the generalization noted in footnote 4, where a 0 delay is allowed between an event and the preceding event in the complete sequence. If this occurs, it means that multiple events have fallen at the same time—yet they still have a well-defined order in which they are generated and read by the LSTM.

An unobserved event may have a 0 delay, if line 33 proposes $\Delta = 0$ and the proposal is accepted. The neural Hawkes model can in principle make such a proposal, but it has zero probability. However, it might have positive probability under a slightly different model.

An observed event may also have a 0 delay, if $t = t_{i+1}$ at line 35 and the proposal is accepted.[16] In this way, it is possible for the proposal distribution to propose any number of unobserved events at time t_{i+1} and immediately before the actual observed event $k_{i+1}@t_{i+1}$. However, once the proposal distribution happens to propose $\Delta > 0$, the actual observed event $k_{i+1}@t_{i+1}$ will preempt the proposal, ending this sequence of J_i unobserved events.

[15] This assumption could trivially be relaxed to allow it to also depend on the missingness of the preceding events, and/or on the future observed events $\mathcal{F}(t)$.

[16] It may seem improbable to propose $t = t_{i+1}$ exactly, but if $t_i = t_{i+1}$, then proposing an unobserved event between these two observed events is just a case of proposing with 0 delay, as in the previous paragraph.

D. Right-to-Left Continuous-Time LSTM

Here we give details of the right-to-left LSTM from section 3.2. Note that this set of formulas is nearly the same as that of Appendix B.1—after all, it is a continuous-time LSTM that has the same architecture as the one in the neural Hawkes process. The difference is that it reads only the observed events, and does so from right to left. The two LSTMs do not share parameters.

The hidden state $\bar{\mathbf{h}}(t)$ is continually obtained from the memory cells $\mathbf{c}(t)$ as the cells decay (in reversed time):

$$\bar{\mathbf{h}}(t) = \mathbf{o}_i \odot (2\sigma(2\bar{\mathbf{c}}(t)) - 1) \text{ for } t \in [t_{i-1}, t_i) \tag{23}$$

where the interval $[t_{i-1}, t_i)$ has consecutive observations $k_{i-1}@t_{i-1}$ and $k_i@t_i$ as endpoints. At t_i, the continuous-time LSTM reads $k_i@t_i$ and updates the current (decayed) hidden cells $\bar{\mathbf{c}}(t)$ to new initial values $\bar{\mathbf{c}}_{i-1}$, based on the current (decayed) hidden state $\bar{\mathbf{h}}(t_i)$, as follows:

$$\bar{\mathbf{i}}_{i-1} \leftarrow \sigma\left(\mathbf{W}_{\mathrm{i}}\mathbf{k}_i + \mathbf{U}_{\mathrm{i}}\bar{\mathbf{h}}(t_i) + \mathbf{d}_{\mathrm{i}}\right) \tag{24a}$$
$$\bar{\mathbf{f}}_{i-1} \leftarrow \sigma\left(\mathbf{W}_{\mathrm{f}}\mathbf{k}_i + \mathbf{U}_{\mathrm{f}}\bar{\mathbf{h}}(t_i) + \mathbf{d}_{\mathrm{f}}\right) \tag{24b}$$
$$\bar{\mathbf{z}}_{i-1} \leftarrow 2\sigma\left(\mathbf{W}_{\mathrm{z}}\mathbf{k}_i + \mathbf{U}_{\mathrm{z}}\bar{\mathbf{h}}(t_i) + \mathbf{d}_{\mathrm{z}}\right) - 1 \tag{24c}$$
$$\bar{\mathbf{o}}_{i-1} \leftarrow \sigma\left(\mathbf{W}_{\mathrm{o}}\mathbf{k}_i + \mathbf{U}_{\mathrm{o}}\bar{\mathbf{h}}(t_i) + \mathbf{d}_{\mathrm{o}}\right) \tag{24d}$$

$$\bar{\mathbf{c}}_{i-1} \leftarrow \bar{\mathbf{f}}_{i-1} \odot \bar{\mathbf{c}}(t_i) + \bar{\mathbf{i}}_{i-1} \odot \bar{\mathbf{z}}_{i-1} \tag{25a}$$
$$\underline{\bar{\mathbf{c}}}_{i-1} \leftarrow \underline{\bar{\mathbf{f}}}_{i-1} \odot \underline{\bar{\mathbf{c}}}_{i} + \underline{\bar{\mathbf{i}}}_{i-1} \odot \bar{\mathbf{z}}_{i-1} \tag{25b}$$
$$\bar{\boldsymbol{\delta}}_{i-1} \leftarrow f\left(\mathbf{W}_{\mathrm{d}}\mathbf{k}_i + \mathbf{U}_{\mathrm{d}}\bar{\mathbf{h}}(t_i) + \mathbf{d}_{\mathrm{d}}\right) \tag{25c}$$

The vector $\mathbf{k}_i \in \{0, 1\}^K$ is the i^{th} input: a one-hot encoding of the new event k_i, with non-zero value only at the entry indexed by k_i. Then, $\bar{\mathbf{c}}(t)$ for $t \in [t_{i-1}, t_i)$ is given by (26), which continues to control $\bar{\mathbf{h}}(t)$ except that i has now decreased by 1.

$$\bar{\mathbf{c}}(t) \stackrel{\text{def}}{=} \underline{\bar{\mathbf{c}}}_{i-1} + \left(\bar{\mathbf{c}}_{i-1} - \underline{\bar{\mathbf{c}}}_{i-1}\right) \exp\left(-\bar{\boldsymbol{\delta}}_{i-1}\left(t_i - t\right)\right) \tag{26}$$

On the interval $[t_{i-1}, t_i)$, $\bar{\mathbf{c}}(t)$ follows an exponential curve that begins at $\bar{\mathbf{c}}_{i-1}$ (in the sense that $\lim_{t \to t_i^-} \bar{\mathbf{c}}(t) = \bar{\mathbf{c}}_{i-1}$) and decays, as time t decreases, toward $\underline{\bar{\mathbf{c}}}_{i-1}$.

E. Optimal Transport Distance Details

Pseudocode is presented in Algorithm 2 for finding optimal transport distance and the corresponding alignment. In the remainder of this section, we prove that optimal transport distance is a valid metric.

It is trivial that OTD is non-negative, since movement, deletion and insertion costs are all positive.

It is also trivial to prove that the following statement is true:

$$L(\mathbf{z}_1, \mathbf{z}_2) = 0 \Leftrightarrow \mathbf{z}_1 = \mathbf{z}_2, \tag{27}$$

where $\mathbf{z}_1$ and $\mathbf{z}_2$ are two sequences. If $\mathbf{z}_1$ is not identical to $\mathbf{z}_2$, the distance of them must be larger than 0 since we have to do some movement, insertion or deletion to make them exactly matched, so the right direction of equation (27) holds. If the distance between $\mathbf{z}_1$ and $\mathbf{z}_2$ is zero, which means they are already matched without any operations, $\mathbf{z}_1$ and $\mathbf{z}_2$ must be identical, thus the left direction of equation (27) holds.

OTD is symmetric, that is, $L(\mathbf{z}_1, \mathbf{z}_2) = L(\mathbf{z}_2, \mathbf{z}_1)$, if we set $C_{\text{insert}} = C_{\text{delete}}$. Suppose that $\mathbf{a}$ is an alignment between $\mathbf{z}_1$ and $\mathbf{z}_2$. It's easy to see that the only difference between $D(\mathbf{z}_1, \mathbf{z}_2, \mathbf{a})$ and $D(\mathbf{z}_2, \mathbf{z}_1, \mathbf{a})$ [17] is that the insertion and deletion operations are exchanged. For example, if we delete a token $t_i \in \mathbf{z}_1$ when calculating $D(\mathbf{z}_1, \mathbf{z}_2, \mathbf{a})$, we should insert a token at t_i to $\mathbf{z}_2$ when calculating $D(\mathbf{z}_2, \mathbf{z}_1, \mathbf{a})$. If we set $C_{\text{insert}} = C_{\text{delete}}$, we have

$$D(\mathbf{z}_1, \mathbf{z}_2, \mathbf{a}) = D(\mathbf{z}_2, \mathbf{z}_1, \mathbf{a}), \quad \forall \mathbf{a} \in \mathcal{A}(\mathbf{z}_1, \mathbf{z}_2). \tag{28}$$

Therefore, we could obtain

$$\begin{aligned} L(\mathbf{z}_1, \mathbf{z}_2) &= \min_{\mathbf{a}^* \in \mathcal{A}(\mathbf{z}_1, \mathbf{z}_2)} D(\mathbf{z}_1, \mathbf{z}_2, \mathbf{a}^*) \\ &= \min_{\mathbf{a}^* \in \mathcal{A}(\mathbf{z}_1, \mathbf{z}_2)} D(\mathbf{z}_2, \mathbf{z}_1, \mathbf{a}^*) = L(\mathbf{z}_2, \mathbf{z}_1) \end{aligned}$$

Finally let's prove that OTD satisfies triangle inequality, that is:

$$L(\mathbf{z}_1, \mathbf{z}_2) + L(\mathbf{z}_2, \mathbf{z}_3) \geq L(\mathbf{z}_1, \mathbf{z}_3), \tag{30}$$

where $\mathbf{z}_1$, $\mathbf{z}_2$ and $\mathbf{z}_3$ are three sequences. This property could be proved intuitively. Suppose that the operations on $\mathbf{z}_1$ with minimal costs to make $\mathbf{z}_1$ matched to $\mathbf{z}_2$ are denoted by $o_1, o_2, \ldots, o_{n_1}$, and those on $\mathbf{z}_2$ to make $\mathbf{z}_2$ matched to $\mathbf{z}_3$ are denoted by $o'_1, o'_2, \ldots, o'_{n_2}$. o_i could be a deletion, insertion or movement on a token. To make $\mathbf{z}_1$ matched to $\mathbf{z}_3$, one possible way, which is not necessarily the optimal, is to do $o_1, o_2, \ldots, o_{n_1}, o'_1, o'_2, \ldots, o'_{n_2}$ on $\mathbf{z}_1$. Since the total cost is the accumulation of the cost of each operation, and the operations on $\mathbf{z}_1$ above to make $\mathbf{z}_1$ matched to $\mathbf{z}_3$ might not be optimal, the triangle inequality equation (30) holds.

F. Approximate MBR Details

Our approximate consensus decoding algorithm is given as Algorithm 3. In the remainder of this section, we prove Theorem 1 from section 4.2, namely:

Theorem 1. *Given* $\{\mathbf{z}_m\}_{m=1}^M$, *if we define* $\mathbf{z}_\sqcup = \bigsqcup_{m=1}^M \mathbf{z}_m$, *then* $\exists \hat{\mathbf{z}} \subseteq \mathbf{z}_\sqcup$ *such that*

$$\textstyle\sum_{m=1}^M w_m L(\hat{\mathbf{z}}, \mathbf{z}_m) = \min_{\mathbf{z} \in \mathcal{Z}} \sum_{m=1}^M w_m L(\mathbf{z}, \mathbf{z}_m)$$

That is to say, there exists one subsequence of $\mathbf{z}_\sqcup$ *that achieves the minimum Bayes risk.*

[17] We abuse the notation $\mathbf{a}$, which we think could represent both the movement from $\mathbf{z}_1$ to $\mathbf{z}_2$ and from $\mathbf{z}_2$ to $\mathbf{z}_1$.

Algorithm 2 A Dynamic Programming Algorithm to Find Optimal Transport Distance

Input: proposal $\hat{\mathbf{z}}$; reference $\mathbf{z}^*$
Output: optimal transport distance d; alignment $\mathbf{a}$

procedure OTD($\hat{\mathbf{z}}, \mathbf{z}^*$)
  $d \leftarrow 0$; $\mathbf{a} \leftarrow$ empty collection $\{\}$
  for $k \leftarrow 1$ **to** K :
    $d^{(k)}, \mathbf{a}^{(k)} \leftarrow \text{ALIGN}(\hat{\mathbf{z}}^{(k)}, \mathbf{z}^{*(k)})$
    $d \leftarrow d + d^{(k)}$; $\mathbf{a} \leftarrow \mathbf{a} \cup \mathbf{a}^{(k)}$
  return $d, \mathbf{a}$
procedure ALIGN($\hat{\mathbf{z}}^{(k)}, \mathbf{z}^{*(k)}$)
  $\hat{I} \leftarrow |\hat{\mathbf{z}}^{(k)}|$; $I^* \leftarrow |\mathbf{z}^{*(k)}|$ ▷ $\hat{\mathbf{z}}^{(k)} = \hat{t}_1, \ldots, \hat{t}_{\hat{I}}$ *and* $\mathbf{z}^{*(k)} = t_1^*, \ldots, t_{I^*}^*$
  $\mathbf{D} \leftarrow$ zero matrix with $(\hat{I}+1)$ rows and (I^*+1) columns
  $\mathbf{P} \leftarrow$ empty matrix with $\hat{I}$ rows and I^* columns ▷ *back pointers*
  for $\hat{i} \leftarrow 1$ **to** $\hat{I}$: ▷ *transport reference of length* 0 *to proposal of length* $\hat{i}$
    $\mathbf{D}_{\hat{i},0} \leftarrow \mathbf{D}_{\hat{i}-1,0} + C_{\text{delete}}$ ▷ *delete* $\hat{t}_{\hat{i}}$ *(and prefixes are matched)*
  for $i^* \leftarrow 1$ **to** I^* : ▷ *transport preference of length* i^* *to proposal of length* 0
    $\mathbf{D}_{0,i^*} \leftarrow \mathbf{D}_{0,i^*} + C_{\text{insert}}$ ▷ *insert* $\hat{t}_{i^*} = t_{i^*}^*$ *to decode (and their prefixes are matched)*
  for $\hat{i} \leftarrow 1$ **to** $\hat{I}$: ▷ *proposal prefix of length* $\hat{i}$
    for $i^* \leftarrow 1$ **to** I^* : ▷ *to match reference of length* i^*
      $D_{\text{delete}} \leftarrow \mathbf{D}_{\hat{i}-1,i^*} + C_{\text{delete}}$ ▷ *if the event token at* $\hat{t}_{\hat{i}}$ *is deleted from* $\hat{\mathbf{z}}^{(k)}$
      $D_{\text{insert}} \leftarrow \mathbf{D}_{\hat{i},i^*-1} + C_{\text{insert}}$ ▷ *if an event token at* $t_{i^*}^*$ *is inserted to* $\hat{\mathbf{z}}^{(k)}$
      $D_{\text{move}} \leftarrow \mathbf{D}_{\hat{i}-1,i^*-1} + |\hat{t}_{\hat{i}} - t_{i^*}^*|$ ▷ *if the event at* $t_{i^*}^*$ *of* $\mathbf{z}^{*(k)}$ *is aligned to event at* $\hat{t}_{\hat{i}}$ *of* $\hat{\mathbf{z}}^{(k)}$
      $\mathbf{D}_{\hat{i},i^*} \leftarrow \min\{D_{\text{insert}}, D_{\text{delete}}, D_{\text{move}}\}$ ▷ *choose the edit that yields the shortest distance*
      $\mathbf{P}_{\hat{i},i^*} \leftarrow \text{argmin}_{e \in \{\text{insert,delete,move}\}} D_e$ ▷ e *represents the kind of edition*
  $\hat{i} \leftarrow \hat{I}$; $i^* \leftarrow I^*$; $\mathbf{a} \leftarrow$ empty collection$\{\}$
  while $\hat{i} > 0$ **and** $i^* > 0$: ▷ *back trace*
    if $\mathbf{P}_{\hat{i},i^*} =$ delete : ▷ *token* $\hat{t}_{\hat{i}}$ *is deleted.*
      $\hat{i} \leftarrow \hat{i} - 1$
    if $\mathbf{P}_{\hat{i},i^*} =$ insert : ▷ *a token at* $t_{i^*}^*$ *is inserted*
      $i^* \leftarrow i^* - 1$
    if $\mathbf{P}_{\hat{i},i^*} =$ move : ▷ *token* $t_{i^*}^*$ *is aligned to* $\hat{t}_{\hat{i}}$
      $\hat{i} \leftarrow \hat{i} - 1; i^* \leftarrow i^* - 1$
      $\mathbf{a} \leftarrow \mathbf{a} \cup \{(\hat{t}_{\hat{i}}, t_{i^*}^*)\}$
  return $\mathbf{D}_{\hat{I},I^*}, \mathbf{a}$

Proof. Here we assume that there is only one type of event. Since the distances of different types of events are calculated separately, our conclusion is easy to be extended to the general case.

Suppose $\hat{\mathbf{z}}$ is an optimal decode, that is,

$$\sum_{m=1}^{M} w_m L(\mathbf{z}_m, \hat{\mathbf{z}}) = \min_{\mathbf{z} \in \mathcal{Z}} \sum_{m=1}^{M} w_m L(\mathbf{z}_m, \mathbf{z}).$$

If $\hat{\mathbf{z}} \subseteq \mathbf{z}_\sqcup$, the proof is done. If not, we can choose some $t_i \notin \mathbf{z}_\sqcup$. Let $t_l = \max\{t \in \mathbf{z}_\sqcup : t < t_i\}$ and $t_r = \min\{t \in \mathbf{z}_\sqcup : t > t_i\}$. (These sets are nonempty because $\mathbf{z}_\sqcup$ always contains the endpoints 0 and T.) We will show that if we move t_i around, as long as $t_i \in [t_l, t_r]$, the weighted optimal transport distance, i.e. $\sum_{m=1}^{M} w_m L(\mathbf{z}_m, \hat{\mathbf{z}})$, will neither increase nor decrease.

Suppose $\hat{\mathbf{a}} = \text{argmin}_{\mathbf{a}_m \in \mathcal{A}(\mathbf{z}_m, \hat{\mathbf{z}})} \sum_{m=1}^{M} w_m D(\mathbf{z}_m, \hat{\mathbf{z}}, \mathbf{a}_m)$. Let's use $r(t)$ to indicate the weighted transport distance of $\hat{\mathbf{z}}$ with fixed alignment if we move t_i to t, that is,

$$r(t) \stackrel{\text{def}}{=} \sum_{m=1}^{M} w_m D(\mathbf{z}_m, \hat{\mathbf{z}}(t), \hat{\mathbf{a}}),$$

where $\hat{\mathbf{z}}(t)$ is the sequence $\hat{\mathbf{z}}$ with t_i moved to t. Because $\hat{\mathbf{z}}(t_i)$ is an optimal decode, and $\hat{\mathbf{a}}$ is the optimal alignment for $\hat{\mathbf{z}}(t_i)$, we should have

$$r(t_i) = \min_t r(t).$$

Note that the transport distance is comprised of three parts: deletion, insertion and alignment costs. Since every $\hat{\mathbf{a}}$ is fixed, if we change t, only the alignment cost that related

Algorithm 3 Approximate Consensus Decoding

Input: collection of weighted particles $\mathcal{Z}_M = \{(\mathbf{z}_m, w_m)\}_{m=1}^M$
Output: consensus sequence $\hat{\mathbf{z}}$ with low $\sum_{m=1}^M w_m L(\hat{\mathbf{z}}, \mathbf{z}_m)$

procedure APPROXMBR($\mathcal{Z}_M$)
  $\hat{\mathbf{z}} \leftarrow$ empty sequence
  for $k = 1$ **to** K :
    $\hat{\mathbf{z}}^{(k)} \leftarrow$ DECODEK($\{(\mathbf{z}_m^{(k)}, w_m)\}_{m=1}^M$); $\hat{\mathbf{z}} \leftarrow \hat{\mathbf{z}} \sqcup \hat{\mathbf{z}}^{(k)}$ ▷ *decode for type-k by calling* DECODEK
  return $\hat{\mathbf{z}}$
procedure DECODEK($\mathcal{Z}_M$)
  ▷ $\mathcal{Z}_M$ *actually means* $\mathcal{Z}_M^{(k)} = \{(\mathbf{z}_m^{(k)}, w_m)\}_{m=1}^M$ *throughout the procedure;* $\mathbf{z}_m$ *is constant*
  $\mathbf{z} \leftarrow \text{argmax}_{\mathbf{z} \in \{\mathbf{z}_m\}_{m=1}^M} w_m$ ▷ *init decode as highest weighted particle and it is global*
  repeat
    for $m = 1$ **to** M : ▷ **Align Phase**
      $d_m, \mathbf{a}_m \leftarrow$ ALIGN($\mathbf{z}, \mathbf{z}_m$) ▷ *call method in Algorithm 2;* $d_m, \mathbf{a}_m$ *are global*
    $r_{\text{min}} \leftarrow \sum_m w_m d_m$ ▷ *track the risk of current* $\mathbf{z}$
    $\mathbf{z}, \{d_m, \mathbf{a}_m\}_{m=1}^M \leftarrow$ MOVE($\mathbf{z}, \{\mathbf{z}_m, d_m, \mathbf{a}_m\}_{m=1}^M$) ▷ *see Algorithm 4*
    $\mathbf{z}, \{d_m, \mathbf{a}_m\}_{m=1}^M \leftarrow$ DELETE($\mathbf{z}, \{\mathbf{z}_m, d_m, \mathbf{a}_m\}_{m=1}^M$) ▷ *see Algorithm 4*
    $\mathbf{z}, \{d_m, \mathbf{a}_m\}_{m=1}^M \leftarrow$ INSERT($\mathbf{z}, \{\mathbf{z}_m, d_m, \mathbf{a}_m\}_{m=1}^M$) ▷ *see Algorithm 4*
  until $\sum_{m=1}^M w_m d_m = r_{\text{min}}$ ▷ *risk stops decreasing*
  return $\mathbf{z}$

to token t will affect $r(t)$. This part of $r(t)$ is linear to t, since we have a constraint $t \in [t_l, t_r]$, which guarantees that it will not cross any other tokens in $\mathbf{z}_\sqcup$.

Since $r(t)$ is linear to $t \in [r_l, t_r]$ and $r(t)$ gets minimized at $t_i \in (t_l, t_r)$, we conclude that

$$r(t) = r(t_i) = \text{Const}, \forall t \in [t_l, t_r].$$

Since $r(t)$ is the upper bound of the weighted optimal transport distance $\sum_{m=1}^M w_m L(\mathbf{z}_m, \hat{\mathbf{z}}(t))$, which also gets the same minimal value at $t_i \in (t_l, t_r)$ as $r(t)$, we could conclude that $\forall t \in [t_l, t_r]$:

$$\sum_{m=1}^M w_m L(\mathbf{z}_m, \hat{\mathbf{z}}(t)) = \sum_{m=1}^M w_m L(\mathbf{z}_m, \hat{\mathbf{z}}(t_i)) = \text{Const}$$

Therefore we could move token t_i to either t_l or t_r without increasing the Bayes risk. We could do this movement for each $t_i \notin \mathbf{z}_\sqcup$ to get a new decode $\hat{\mathbf{z}} \subseteq \mathbf{z}_\sqcup$, which is also an optimal decode.

□

G. Experimental Details

In this section, we elaborate on the details of data generation, processing, and experimental results.

In all of our experiments, the distribution p_{model} is trained on the complete (uncensored) version of the training data. The system is then asked to complete the incomplete (censored) version of the test (or dev) data. For particle smoothing, the proposal distribution is trained using both the complete and incomplete versions of the training data, as explained at the end of section 3.2.1. We used the Adam algorithm with its default settings (Kingma & Ba, 2015). Adam is a stochastic gradient optimization algorithm that continually adjusts the learning rate in each dimension based on adaptive estimates of low-order moments. Each training example for Adam is a complete event stream $\mathbf{x} \sqcup \mathbf{z}$ over some time interval $[0, T)$. We stop training early when we detect that log-likelihood has stopped increasing on the held-out development dataset. We do no other regularization.

G.1. Dataset Statistics

Table 1 shows statistics about each dataset that we use in this paper.

G.2. Training Details

We used single-layer LSTMs (Hochreiter & Schmidhuber, 1997), selected the number D of hidden nodes of the left-to-right LSTM, and then D' of the right-to-left one from a small set $\{16, 32, 64, 128, 256, 512, 1024\}$ based on the performance on the dev set of each dataset. The best-performing (D, D') pairs are $(256, 128)$ on Synthetic, $(256, 256)$ on Elevator $(256, 256)$ on NYC Taxi, but we empirically found that the model performance is robust to these hyperparameters. For the chosen (D, D') pair on each dataset, we selected β based on the performance on the dev set, and $\beta = 1.0$ yields the best performance across all the datasets we use. For learning, we used Adam with

Algorithm 4 Subroutines for Approximate Consensus Decoding

procedure MOVE($\mathbf{z}, \{\mathbf{z}_m, d_m, \mathbf{a}_m\}_{m=1}^M$) ▷ **Move Phase**
  for t **in** $\mathbf{z}$:
    for $t' \in \{t' : (t', t) \in \bigcup_{m=1}^M \mathbf{a}_m\}$: ▷ *may replace t with t' which is aligned to t*
      $(\forall m) d'_m \leftarrow d_m$
      for $(t'', m) \in \{(t'', m) : (t'', t) \in \mathbf{a}_m, m \in \{1, \dots, M\}\}$:
        $d'_m \leftarrow d'_m - |t'' - t| + |t'' - t'|$
      if $\sum_m w_m d'_m < \sum_m w_m d_m$:
        $(\forall m) d_m \leftarrow d'_m$; $t \leftarrow t'$ ▷ *t move to t' for lower risk*
  return $\mathbf{z}, \{d_m, \mathbf{a}_m\}_{m=1}^M$
procedure DELETE($\mathbf{z}, \{\mathbf{z}_m, d_m, \mathbf{a}_m\}_{m=1}^M$) ▷ **Delete Phase**
  for t **in** $\mathbf{z}$: ▷ *may delete this event*
    for $m = 1$ **to** M : ▷ *update each d_m*
      if $\exists t' \in \mathbf{z}_m$ **and** $(t', t) \in \mathbf{a}_m$: ▷ *find the only, if any, $t' \in \mathbf{z}_m$ that is aligned to t*
        ▷ *if we delete t and its alignment (t', t), d_m decreases by the alignment cost (because we do not need to align it)*
        ▷ *but increases by an insertion cost (because we need to insert an event at t to match $\mathbf{z}_m$)*
        $d'_m \leftarrow d_m + C_{\text{insert}} - |t' - t|$
      else ▷ *otherwise, this event has been deleted when matching with $\mathbf{z}_m$*
        $d'_m \leftarrow d_m - C_{\text{delete}}$ ▷ *we do not need to pay deletion cost when matching with $\mathbf{z}_m$ if we do not have this event at t in $\mathbf{z}$*
      if $\sum_m w_m d'_m < \sum_m w_m d_m$:
        delete t from $\mathbf{z}$; $(\forall m)$ delete (t', t) from $\mathbf{a}_m$; $d_m \leftarrow d'_m$
  return $\mathbf{z}, \{d_m, \mathbf{a}_m\}_{m=1}^M$
procedure INSERT($\mathbf{z}, \{\mathbf{z}_m, d_m, \mathbf{a}_m\}_{m=1}^M$) ▷ **Insert Phase**
  repeat
    $t \leftarrow$ None, $\Delta \leftarrow -\infty$
    for $t_c \in \bigsqcup_m \mathbf{z}_m$ such that $t_c \notin \mathbf{z}$: ▷ *may insert t_c if it is not in $\mathbf{z}$ yet*
      for $m = 1$ **to** M :
        $\mathbf{z}'_m \leftarrow \{t' : \forall t'', (t'', t') \notin \mathbf{a}_m \text{ and } t' \in \mathbf{z}_m\}$ ▷ *find t' in $\mathbf{z}_m$ that is not aligned yet*
        if $\mathbf{z}'$ is not empty **and** $\min_{t' \in \mathbf{z}'_m} |t' - t_c| < C_{\text{insert}} + C_{\text{delete}}$: ▷ *if there is any that is close enough to t_c*
          $d'_m \leftarrow d_m - C_{\text{insert}} + \min_{t' \in \mathbf{z}'_m} |t' - t_c|$; $\mathbf{a}'_m \leftarrow \mathbf{a}_m \cup \{(t_c, t')\}$ ▷ *align the closest one to t_c*
        else
          $d'_m \leftarrow d_m + C_{\text{delete}}$; $\mathbf{a}'_m \leftarrow \mathbf{a}_m$
      if $\sum_m w_m d_m - \sum_m w_m d'_m > \Delta$:
        $t \leftarrow t_c$; $\Delta \leftarrow \sum_m w_m d_m - \sum_m w_m d'_m$
    if $\Delta > 0$:
      $\mathbf{z} \leftarrow \mathbf{z} \sqcup \{t\}$; $(\forall m) \mathbf{a}_m \leftarrow \mathbf{a}'_m$; $d_m \leftarrow d'_m$
  until $\Delta \leq 0$
  return $\mathbf{z}, \{d_m, \mathbf{a}_m\}_{m=1}^M$

DATASET	K	# OF EVENT TOKENS			SEQUENCE LENGTH		
		TRAIN	DEV	TEST	MIN	MEAN	MAX
SYNTHETIC	4	≈ 74967	≈ 7513	≈ 7507	10	≈ 15	20
NYCTAXI	10	157916	15826	15808	22	32	38
ELEVATOR	10	313043	31304	31206	235	313	370

Table 1. Statistics of each dataset. We write "$\approx N$" to indicate that N is the average value over multiple datasets of one kind (synthetic); the variance is small in each such case.

its default settings (Kingma & Ba, 2015).

Our Monte Carlo integral estimates are in fact unbiased (Appendix C.3). As a result, our stochastic gradient estimate is also unbiased, as required (assuming that the complete data is distributed according to p_{model}). Why? Since $\beta = 1$, our stochastic gradient is simply equation (6). No particle filtering or smoothing is used to estimate equation (6), because we train it using complete data, as explained in the last long paragraph of section 3.2.1. The only randomness is the integral over $[0, T)$ (similar to the one in equation (19)) that is required to estimate the term $\log q(\mathbf{z} \mid \mathbf{x})$ in equation (6): as just noted, this integral estimate is unbiased.

It is true that if $\beta < 1$, we would compute the exclusive KL gradient using particle filtering or smoothing with M particles, and this would introduce bias in the gradient. Nonetheless, since the bias vanishes as $M \to \infty$, it would be possible to restore a theoretical convergence guarantee by increasing M at an appropriate rate as SGD proceeds (Spall, 2005, page 107).[18]

G.3. Details of the Synthetic Datasets

Each of the ten neural Hawkes processes has its parameters sampled from $\text{Unif}[-1.0, 1.0]$. Then a set of event sequences is drawn from each of them via the plain vanilla thinning algorithm (Mei & Eisner, 2017). For each of the ten synthetic datasets, we took $K = 4$ as the number of event types. To draw each event sequence, we first chose the sequence length I (number of event tokens) uniformly from $\{11, 12, \ldots, 20\}$ and then used the thinning algorithm to sample the first I events over the interval $[0, \infty)$. For subsequent training or testing, we treated this sequence (appropriately) as the complete set of events observed on the interval $[0, T)$ where $T = t_I$, the time of the last generated event.

We generate 5000, 500 and 500 sequences for each training, dev, and test set respectively. For the missingness mechanism: in the deterministic settings, we censor all events of type 3 and 4—in other words, we set $\rho_1 = \rho_2 = 0$ and $\rho_3 = \rho_4 = 1$; in the stochastic settings, we set $\rho_k = 0.5$ for all k.

G.4. Elevator System Dataset Details

We examined our method in a simulated 5-floor building with 2 elevator cars. During a typical afternoon down-peak rush hour (when passengers go from floor-2,3,4,5 down to the lobby), elevator cars travel to each floor and pick up passengers that have (stochastically) arrived there according to a traffic profile (Bao et al., 1994). Each car will also avoid floors that already are or will soon be taken care of by the other. Having observed when and where car-1 has stopped (to pick up or drop off passengers) over this hour, we are interested in when and where car-2 has stopped during the same time period. In this dataset, each event type is a tuple of (car number, floor number) so there are $K = 10$ in total in this simulated 5-floor building with 2 elevator cars.

Passenger arrivals at each floor are assumed to follow a inhomogeneous Poisson process, with arrival rates that vary during the course of the day. The simulations we use follows a human-recorded traffic profile (Bao et al., 1994) which dictates arrival rates for every 5-minute interval during a typical afternoon down-peak rush hour. Table 2 shows the mean number of passengers (who are going to the lobby) arriving at floor-2,3,4,5 during each 5-minute interval.

We simulated the elevator behavior following a naive baseline strategy documented in Crites & Barto (1996).[19] In details, each car has a small set of primitive actions. If it is stopped at a floor, it must either "move up" or "move down". If it is in motion between floors, it must either "stop at the next floor" or "continue past the next floor". Due to passenger expectations, there are two constraints on these actions: a car cannot pass a floor if a passenger wants to get off there and cannot turn until it has serviced all the

[18] SGD methods succeed, both theoretically and practically, with even high-variance estimates of the batch gradient (e.g., where each stochastic estimate is derived from a single randomly chosen training example). Thus, one should be fine with a noisy sampling-based gradient as long as it is unbiased.

[19] We rebuilt the system in Python following the original Fortran code of Crites & Barto (1996).

car buttons in its current direction. Three additional action constraints were made in an attempt to build in some primitive prior knowledge: 1) a car cannot stop at a floor unless someone wants to get on or off there; 2) it cannot stop to pick up passengers at a floor if another car is already stopped there; 3) given a choice between moving up and down, it should prefer moving up (since the down-peak traffic tends to push the cars toward the bottom of the building). Because of this last constraint, the only real choices left to each car are the stop and continue actions, and the baseline strategy always chooses to continue. The actions of the elevator cars are executed asynchronously since they may take different amounts of time to complete.

We repeated the (one-hour) simulation 700 times to collect the event sequences, each of which has around 300 time-stamped records of which car stops at which floor. We randomly sampled disjoint train, dev and test sets with 500, 100 and 100 sequences respectively.

For the missingness mechanism: in the deterministic settings, we set $\rho_k = 0$ for $k = 1, \ldots, 5$ and $\rho_k = 1$ for $k = 6, \ldots, 10$ (meaning that the events (of arriving at floor 1, 2, ..., 5) of car 1 are all observed, but those of car 2 are not); in the stochastic settings, we set $\rho_k = 0.5$ for all k.

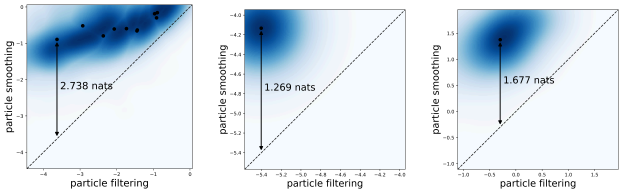
G.5. New York City Taxi Dataset Details

The New York City Taxi dataset (section 5.2) includes 189,550 taxi pick-up and drop-off records in the city of New York in 2013. Each record has its medallion ID, driver license and time stamp. Each combination of medallion ID and driver license naturally forms a sequence of time-stamped pick-up and drop-off events. Following the processing recipe of previous work (Du et al., 2016), we construct shorter sequences by breaking each long sequence wherever the temporal gap between a drop-off event and its following pick-up event is larger than six hours. Then the left boundary of this gap is treated as the EOS of the sequence before it, while the right boundary is set as the BOS of the following sequence.

We randomly sampled a month from 2013 and then randomly sampled disjoint train, dev and test sets with 5000, 500 and 500 sequences respectively from that month.

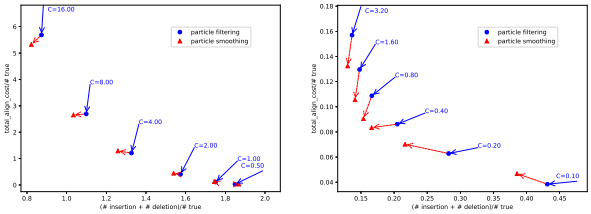
In this dataset, each event type is a tuple of (location, action). The location is one of the 5 boroughs {Manhattan, Brooklyn, Queens, The Bronx, Staten Island}. The action can be either pick-up or drop-off. Thus, there are $K = 5 \times 2 = 10$ event types in total.

For the missingness mechanism: in the deterministic settings, we set $\rho_k = 0$ for $k = 1, \ldots, 5$ and $\rho_k = 1$ for $k = 6, \ldots, 10$ (which means that all drop-off events but no pick-up events are observed); in the stochastic settings, we set $\rho_k = 0.5$ for all k.



(a) Synthetic (b) Elevator System (c) NYC Taxi

Figure 4. Scatterplots with a deterministic missingness mechanism. Again, the method works, with very similar qualitative behavior to Figure 2.



(a) Elevator System (b) NYC Taxi

Figure 5. Optimal transport distance results with a deterministic missingness mechanism. Again, the method works, with very similar qualitative behavior to Figure 3.

G.6. Experiments with Deterministic Missingness Mechanisms

We show our experimental results for the deterministic missingness mechanisms in Figures 4 and 5.

G.7. Sensitivity Experiment Details

Figure 6 displays the optimal transport distance with various values of ρ: our particle smoothing method consistently outperforms the filtering baseline.

G.8. Wall-Clock Runtime Details

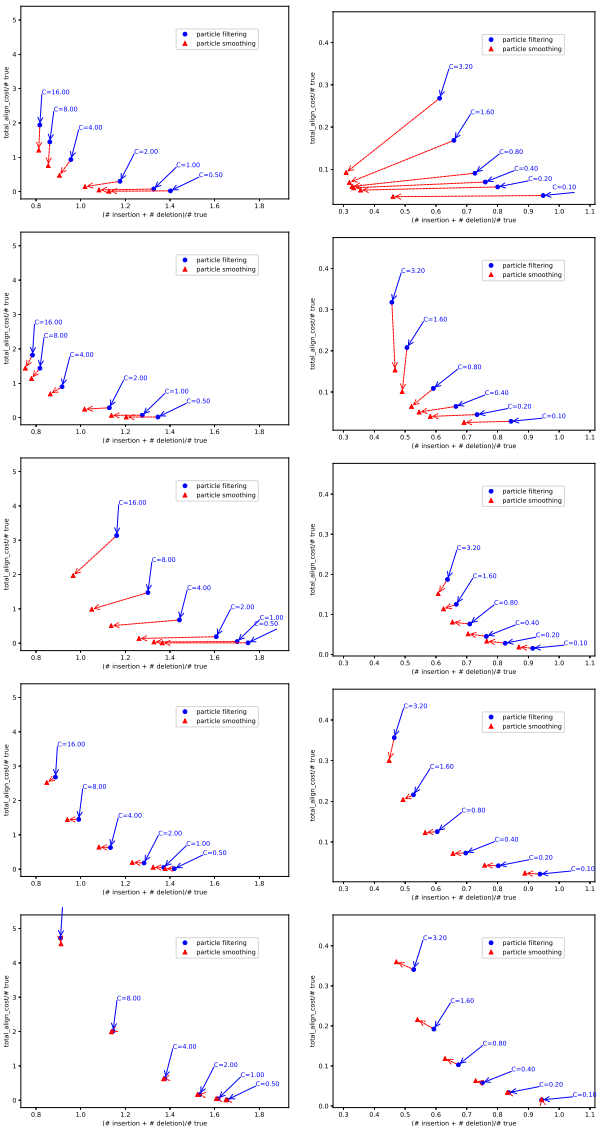
A given run of particle smoothing begins by drawing $O(I)$ time points from $\mathrm{Unif}([0, T))$, where I is the number of observed events. All particles are evaluated using integrals that are estimated by evaluating the function at these time points (Appendix C.3).

The theoretical runtime complexity is $O(MI)$ because drawing a particle requires $O(I)$ time—the outer loop over time steps (line 9 of Algorithm 1)—and we draw M particles in total—the inner loop over particles (line 10 in Algorithm 1). Our GPU implementation (which we will release) parallelizes the inner loop over particles. We sample 50 particles in parallel in these experiments, but we have tested with 1000 particles in parallel as well. So this is not a real problem with a GPU.

We reported experiments that we performed to demon-

START TIME (MIN)	00	05	10	15	20	25	30	35	40	45	50	55
MEAN # PASSENGER	1	2	4	4	18	12	8	7	18	5	3	2

Table 2. The Down-Peak Traffic Profile



(a) Elevator System (b) NYC Taxi

Figure 6. Optimal transport distance results with varying missingness rate ρ. Rows (top-down) are results with $\rho = 0.1, 0.3, 0.5, 0.7, 0.9$. As we can see, our particle smoothing consistently outperforms the filtering baseline with different ρ, although no clear trend with increasing ρ is found on either dataset.

strate the practicality. On average, drawing an ensemble of 50 particles takes 5 seconds per example on the synthetic datasets (average length 15 events), 12 seconds per example on the NYC Taxi dataset (average length 32 events) and 100 seconds per example on the Elevator System dataset (average length 313 events)—that is, 300-400 milliseconds per event. Such speeds are acceptable in many incomplete data applications, compared to the cost of collecting complete data—all the applications in section 1 involve real-time decision making at a human timescale.

H. Monte Carlo EM

We normally assume (section 3.2.1) that some complete sequences are available for training the neural Hawkes process models. If incomplete sequences are also available, our particle smoothing method can be used to (approximately) impute the missing events, which yields additional complete sequences for training. Indeed, if we are willing to make a MAR assumption (Little & Rubin, 1987), then we can do imputation without modeling the missingness mechanism. Training on such imputed sequences is an instance of **Monte Carlo expectation-maximization (MCEM)** (Dempster et al., 1977; Wei & Tanner, 1990; McLachlan & Krishnan, 2007), with particle smoothing as the Monte Carlo E-step, and makes it possible to train with incomplete data only.

In the more general MNAR scenario, we can extend the E-step to consider the not-at-random missingness mechanism (see equation (3) below), but then we need both complete and incomplete sequences at training time in order to fit the parameters of the missingness mechanism (unless these parameters are already known) jointly with those of the neural Hawkes process. Although training with incomplete data is out of the scope of our experiments, we describe the methods here and provide MCEM pseudocode.

In this case, we would like to know the (marginal) probability of the observed data $\mathbf{x}$ under the target distribution p:

$$p(\mathbf{x}) = \sum_{\mathbf{z}} p_{\text{model}}(\mathbf{x} \sqcup \mathbf{z}) p_{\text{miss}}(\mathbf{z} \mid \mathbf{x} \sqcup \mathbf{z}) \tag{31}$$

If we propose $\mathbf{z}$ from $q(\mathbf{z} \mid \mathbf{x})$, then it can be rewritten as:

$$p(\mathbf{x}) = \sum_{\mathbf{z}} p_{\text{model}}(\mathbf{x} \sqcup \mathbf{z}) p_{\text{miss}}(\mathbf{z} \mid \mathbf{x} \sqcup \mathbf{z}) \frac{q(\mathbf{z} \mid \mathbf{x})}{q(\mathbf{z} \mid \mathbf{x})} \tag{32a}$$

$$= \mathbb{E}_{\mathbf{z} \sim q(\mathbf{z}|\mathbf{x})} \left[\frac{p_{\text{model}}(\mathbf{x} \sqcup \mathbf{z}) p_{\text{miss}}(\mathbf{z} \mid \mathbf{x} \sqcup \mathbf{z})}{q(\mathbf{z} \mid \mathbf{x})} \right] \tag{32b}$$

Given a finite number M of proposed particles $\{\mathbf{z}_m\}_{m=1}^M$, this expectation can be estimated with empirical average:

$$p(\mathbf{x}) = \frac{1}{M} \sum_{m=1}^{M} \frac{p_{\text{model}}(\mathbf{x} \sqcup \mathbf{z}_m) p_{\text{miss}}(\mathbf{z}_m \mid \mathbf{x} \sqcup \mathbf{z}_m)}{q(\mathbf{z}_m \mid \mathbf{x})} \tag{33}$$

and it is obvious that

$$\log p(\mathbf{x}) \geq \frac{1}{M} \sum_{m=1}^{M} (b_m - \log q(\mathbf{z}_m \mid \mathbf{x})) \tag{34a}$$

$$b_m = \log p_{\text{model}}(\mathbf{x} \sqcup \mathbf{z}_m) + \log p_{\text{miss}}(\mathbf{z}_m \mid \mathbf{x} \sqcup \mathbf{z}_m) \tag{34b}$$

where the right-hand-side (RHS) term of equation (34a) is the **Evidence Lower Bound (ELBO)** that we would maximize in order to maximize the log-likelihood.

The MCEM algorithm is composed of two steps:

E(xpectation)-step We train the proposal distribution $q(\mathbf{z} \mid \mathbf{x})$ using the method in section 3.2.1 and then sample M weighted particles from $q(\mathbf{z} \mid \mathbf{x})$ by calling Algorithm 1.

M(aximization)-step We train the neural Hawkes process $p_{\text{model}}(\mathbf{x} \sqcup \mathbf{z})$ by maximizing the ELBO (equation (34a)).

Note that in the MAR case, $p_{\text{miss}}(\mathbf{z} \mid \mathbf{x} \sqcup \mathbf{z})$ is constant of $\mathbf{z}$ so the it can be omitted from the formulation (and thus the algorithms). Also note that, for particle filtering, the proposal distribution $q(\mathbf{z} \mid \mathbf{x})$ is only part of $p_{\text{model}}(\mathbf{x} \sqcup \mathbf{z})$ so we do not need to train $q(\mathbf{z} \mid \mathbf{x})$ at the E-step.

Maximum-likelihood estimation remains sensible in the MNAR case provided that we know one of the distributions p_{model} or p_{miss}, in which case we can use EM to estimate the other distribution.

(1) If p_{miss} is known and fixed, as in our experiments, this gives a minor variant of ordinary EM. Ordinary EM makes the MAR assumption that the p_{miss} factor of equation (1) can be ignored. However, if we know p_{miss}, we can incorporate it rather than ignoring it; then it need not satisfy the MAR assumption.

(2) Conversely, if p_{model} is known and fixed because we estimated it from a sufficient quantity of *complete* data, then we can use incomplete data to learn the MNAR missingness distribution p_{miss}. This setting would even lets us learn contextual missingness mechanisms in which the probability that an event is censored depends not only on the event itself, but also on the surrounding events and whether they are censored. For example, one could try to fit p_{miss} with an LSTM model or a BiLSTM-CRF model (Huang et al., 2015) that performs structured joint prediction of the missingness of all events in the sequence. Extending that method to use continuous-time LSTMs would allow it to take timing into account.

The E step of Monte Carlo EM uses the current guesses of p_{model} and/or p_{miss} to sample from the posterior distribution $p(\mathbf{z} \mid \mathbf{x})$ of the missing values. That posterior is uncontroversially defined by the simple Bayesian formula (1). Notice that even if p_{model} and p_{miss} were *both* unknown, we could still run MCEM to locally maximize the likelihood $p(\mathbf{x})$, but unfortunately the parameters would be unidentifiable in this case. Thus, there would be many missing-data models with the same likelihood, as explained in Appendix A, and they would make different predictions of $\mathbf{z}$.