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 $z$. Suppose we know the prior distribution $p_{\text{model}}$ of complete event streams, as well as the “missingness mechanism” $p_{\text{miss}}(z \mid \text{complete stream})$, which stochastically determines which of the events will not be observed. One can then use Bayes’ Theorem, as spelled out in equation (1) below, to define the posterior distribution $p(z \mid x)$ given just the observed events $x$.

Why is it important? The ability to impute $z$ 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 $p(z \mid x)$. Even for a simple $p_{\text{model}}$ 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.

Imputing Missing Events in Continuous-Time Event Streams

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 $t_2$.

(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 □ at time $t_2$ into the LSTM and proposing subsequent events based on the new intensities after $t_2$. But because ( ) was low at $t_2$, the ( ) 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 (■) far apart in time.

(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. 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).

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

### 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 $k_{\text{at}}$. 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 $x = \{k_1\text{at}_1, k_2\text{at}_2, \ldots, k_I\text{at}_I\}$, 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** $k_0\text{at}_0$ and $k_{I+1}\text{at}_{I+1}$ at its endpoints, where $k_0 = 0$ and $k_{I+1} = K+1$.

Let $k_{i,0}\text{at}_i$ be an alternative notation for the observed event $k_i\text{at}_i$. Following this observed event (for any $0 \leq i \leq I$), there are $J_i \geq 0$ **unobserved events** $z = \{k_{i,1}\text{at}_{i,1}, k_{i,2}\text{at}_{i,2}, \ldots, k_{i,J_i}\text{at}_{i,J_i}\}$, 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** $x \sqcup z$ is thus $\{k_{i,j}\text{at}_{i,j} : 0 \leq i \leq I+1, 0 \leq j \leq J_i\}$, where $t_{i,j}$ increases strictly with the pair $(i, j)$ in lexicographic order.

---

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

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

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

of a process that first generates the complete sequence $x \sqcup z$ from a complete data model $p_{\text{model}}$ (given $[0, T]$), and then determines which events to censor with the possibly stochastic **missingness mechanism** $p_{\text{miss}}$. 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 $z$ denotes the set of missing events in $Y$ and $Z = z$ denotes the fact that they are missing. That said, we will abbreviate our notation above in the standard way:

$$p(z \mid x) \propto p_{\text{model}}(x \sqcup z) \cdot p_{\text{miss}}(z \mid x \sqcup z)$$  \hspace{1cm} (1)$$

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

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

### 2.2. Choice of $p_{\text{model}}$

We need a multivariate point process model $p_{\text{model}}(x \sqcup z)$. 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 $\mathcal{H}(t) \overset{\text{def}}{=} \{k'\text{at} : t' < t\}$—the set of all observed and unobserved events before $t$. Given this history, the neural Hawkes process defines an **intensity** $\lambda_k(t \mid \mathcal{H}(t)) \in \mathbb{R}_{>0}$, which may be thought of as the instantaneous rate at time $t$ of events of type $k$:

$$\lambda_k(t \mid \mathcal{H}(t)) = f_k(v_k^\top \mathbf{h}(t))$$  \hspace{1cm} (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, $\lambda_k$ is similar to the intensity function of an inhomogeneous Poisson process. Yet it is not a fixed parameter: the $\lambda_k$ 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 $p(z \mid x)$, because $x$ and $z$ can be interleaved with each other. As an alternative, we can use normalized importance sampling, drawing many $z$ values from a proposal distribution $q(z \mid x)$ and weighting them in proportion to $\frac{p(z \mid x)}{q(z \mid x)}$. 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 $Z_M = \{(z_m, w_m)\}_{m=1}^M$. Each particle $z_m$ is sampled from the proposal distribution $q(z \mid x)$, 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):

\[
w_m \propto \frac{p_{\text{model}}(\bar{x} \cup z_m) p_{\text{miss}}(z_m \mid \bar{x} \cup z_m)}{q(z_m \mid \bar{x})} \geq 0 \tag{3}
\]

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 $q(z \mid x)$ equal to $p(z \mid x)$, 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, $\bar{p}(z)$, approaches $p(z \mid x)$ as $M \to \infty$ (Doucet & Johansen, 2009). Thus, for large $M$, the ensemble may be used to estimate the expectation of any function $f(z)$, via

\[
\mathbb{E}_{p(z \mid x)}[f(z)] \approx \mathbb{E}_{\bar{p}}[f(z)] = \sum_{m=1}^M w_m f(z_m) \tag{4}
\]

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

In the subsections below, we will describe two specific proposal distributions $q$ that are appropriate for the neural Hawkes process, as we sketches in section 1. These distributions define intensity functions $\lambda^n$ 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 $\lambda^*$, 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 $\lambda^*(t)/\lambda^* \leq 1$. 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 $\{z_m\}_{m=1}^M$ (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 $p_{\text{model}}$ that was trained on complete data. This model uses a neural net to define an intensity function $\lambda^n_k(t \mid \mathcal{H}(t))$ 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 $t \in (t_{i,j}, t_{i,j+1})$, 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 $\text{smooth} = \text{false}$.

### 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 $\mathcal{F}(t) \triangleq \{k_{i,j}@t_i : t < t_i \leq T\}$ 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 $h(t) \in (-1, 1)^D$. Then we parameterize the proposal intensity using an extended variant of equation (2):

\[
\lambda^n_k(t \mid \mathcal{H}(t), \mathcal{F}(t)) = f_k(\vec{h}(t) + B h(t)) \tag{5}
\]

This extra machinery is used by Algorithm 1 when $\text{smooth} = \text{true}$. Intuitively, the left-to-right $h(t)$, 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 $h(t)$, 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 $p_{\text{model}}$ 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 $p_{\text{model}}$, 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 $p_{\text{model}}$ (section 2.2), but separate parameter vectors. For any time $t \in [0, T)$, it arrives at $\mathbf{h}(t)$ by reading only the observed events $\{k_t, \mathbf{u}_t : t < t_i \leq T\}$, 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 $p(z \mid x)$ by minimizing a Kullback-Leibler (KL) divergence. Its left-to-right LSTM is fixed at $p_{\text{model}}$, so its trainable parameters $\phi$ are just the parameters of the right-to-left LSTM together with the matrix $\mathbf{B}$ from equation (5). Though $p(z \mid x)$ is unknown, the gradient of inclusive KL divergence between $q(z \mid x)$ and $p(z \mid x)$ is

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

and the gradient of exclusive KL divergence is:

$$\nabla_\phi \text{KL}(q \parallel p) = \mathbb{E}_{z \sim \cdot \cdot \cdot q(z \mid x)}\left[\nabla_\phi \left(\frac{1}{2} \log q(z \mid x) - b)\right)\right]$$

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

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

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

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

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

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

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

$$\hat{z} = \arg\min_{z \in \mathcal{Z}} \sum_{z^\ast \in \mathcal{Z}} p(z^\ast \mid x) L(z, z^\ast)$$

$$\approx \arg\min_{z \in \mathcal{Z}} \sum_{m=1}^{M} w_m L(z, z_m) \quad \text{(9a)}$$

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

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

3 To get more data for training $q$, we could sample more partials 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 \( A(z, z^*) \) is the set of all possible alignments between \( z \) and \( z^* \), and \( D(z, z^*, a) \) is the total cost given the alignment \( a \). Notice that if \( |z| \neq |z^*| \), 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 \( a \), 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 \( \hat{z}^{(k)} \) of type-\( k \) events for each \( k = 1, 2, \ldots, K \), based on the particles’ sets \( z^{(k)}_m \) 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 \( \{z_m\}_{m=1}^M \), if we define \( \mathcal{z}_{\downarrow} = \bigcup_{m=1}^M z_m \), then \( \exists \hat{z} \subseteq \mathcal{z}_{\downarrow} \) such that

\[
\sum_{m=1}^M w_m L(\hat{z}, z_m) = \min_{z \in \mathcal{z}_{\downarrow}} \sum_{m=1}^M w_m L(z, z_m)
\]

That is to say, there exists a subsequence of \( \mathcal{z}_{\downarrow} \) that achieves the minimum Bayes risk.

The proof is given in Appendix F: it shows that if \( \hat{z} \) minimizes the Bayes risk but is not a subsequence of \( \mathcal{z}_{\downarrow} \), 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 \( \mathcal{z}_{\downarrow} \) as desired.

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

\[
\hat{z} = \arg\min_{z \subseteq z_{\downarrow}} \sum_{m=1}^M w_m L(z, z_m)
\]

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 \( \hat{z} \) by (1) using Algorithm 2 to find the optimal alignment \( a_m \) of \( \hat{z} \) with each \( z_m \), and then (2) repeating the following sequence of 3 phases until \( \hat{z} \) does not change. Each phase tries to update \( \hat{z} \) to decrease the weighted distance \( \sum_{m=1}^M w_m D(\hat{z}, z_m, a_m) \) which by Theorem 1 is an upper bound of the Bayes risk \( \sum_{m=1}^M w_m L(\hat{z}, z_m) \).

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

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

**Insert Phase** If we inserted \( t \) into \( \hat{z} \), we would also modify each \( a_m \) to align to the closest unaligned event in \( z_m \) (if any) provided that this decreased \( D(\hat{z}, z_m, a_m) \). Let \( \Delta(t) \) be the resulting reduction in \( \sum_{m=1}^M w_m D(\hat{z}, z_m, a_m) \). Let \( t^* = \arg\max_{t \in z_{\downarrow}, t \notin \hat{z}} \Delta(t) \). While \( \Delta(t^*) \geq 0 \), 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

\[
p_{\text{miss}}(z | x \cup z) = \prod_{k, \delta t \in z} \rho_k \prod_{k, \delta t \in x} (1 - \rho_k)
\]

meaning that each event in the complete stream \( x \cup z \) is independently censored with probability \( \rho_k \) that only depends on its event type \( k \).\(^7\) We consider both deterministic and stochastic missingness mechanisms. For the deterministic experiments, we set \( \rho_k \) for each \( k \) to be either 0 or 1, so that some event types are always observed while others are always missing. Then \( p_{\text{miss}}(z | x \cup z) = 1 \) if \( z \) consists of precisely the events in \( x \cup z \) 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 \( p_{\text{miss}}(z | x \cup z) = (1 - \rho)^{|z|} \rho^{|x|} \), whose value decreases exponentially in the number of missing events \( |z| \). As this depends on \( z \), 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

\( ^7 \)Appendix H discusses how \( \rho \) 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 x part as input to the proposal distribution but we also know the 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 z∗ is under the proposal distribution constructed by each method, i.e., \( \log q(z^* \mid x) \). 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 x, we now make a prediction by sampling an ensemble of \( M = 50 \) particles (section 3)\(^9\) and constructing their consensus sequence \( \hat{z} \) (section 4.2). We use multinomial resampling since otherwise the effective sample size is very low (only 1–2 on some datasets).\(^10\) We evaluate \( \hat{z} \) by its optimal transport distance (section 4.1) to the ground truth \( z^* \). Note that \( \forall a \), we can decompose \( D(\hat{z}, z^*, a) \) as

\[
C \cdot \left( |\hat{z}| + |z^*| - 2|a| \right) + \sum_{(t,t') \in a} |t - t'|
\]

(13)

Letting \( a \) be the alignment that minimizes \( D(\hat{z}, z^*, a) \), the former term measures how well \( \hat{z} \) predicts which events happened, and the latter measures how well \( \hat{z} \) predicts when those events happened. Different choices of \( C \) yield different \( \hat{z} \) 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, \( \hat{z} \) 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.

\(^9\)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.
smoothing method. The transition kernel could propose a single-event change to \( z \) (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 \( O(|x|L|z|) \) 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 \( x \) events, from an approximation \( q(z | x) \) 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 (Hiroswa 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 \( \{A, C, G, T\} \) 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


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.


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