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).

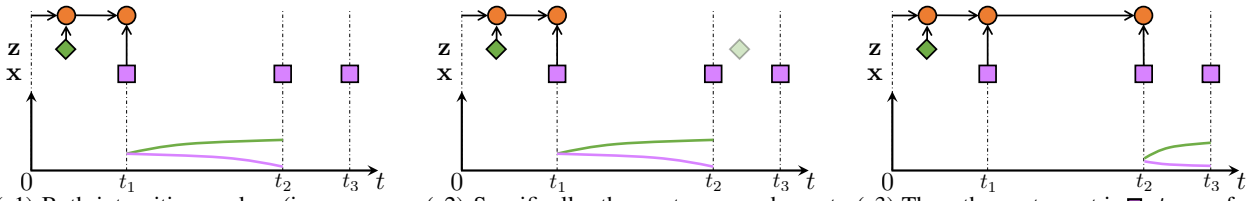
What is our contribution? We are, to the best of our knowledge, the first to develop general sequential Monte Carlo (SMC) methods to approximate the posterior distribution over incompletely observed draws from a neural point process. We begin by sketching the approach.

[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.

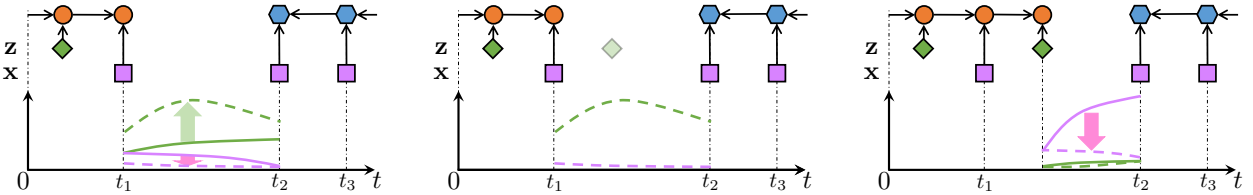


(a1) Both intensities are low (i.e., passengers are scarce at this time of day), so no event happens to be proposed in (t_1, t_2).

(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.

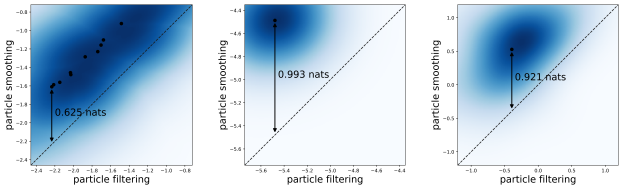
New York City Taxi Dataset (Whong, 2014)**.** Each medallion taxi in New York City has a sequence of time-stamped pick-up and drop-off events, where different locations have different event types. Figure 1 shows how we impute the pick-up events given the drop-off events (the deterministic missingness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

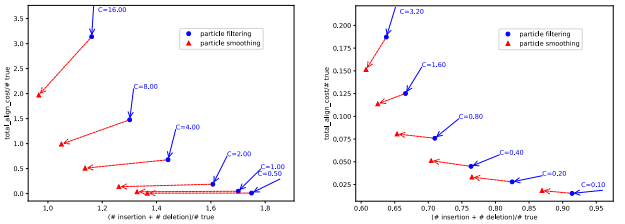
For the stochastic missingness mechanisms, we also did experiments with different values of missing rate $\rho = 0.1, 0.3, 0.7, 0.9$. Our particle smoothing method consistently outperforms the filtering baseline in all the experiments (Figure 6 in Appendix G.7), similar to Figure 3.

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.