

# The Neural Hawkes Process: A Neurally Self-Modulating Multivariate Point Process

Hongyuan Mei    Jason Eisner

Department of Computer Science, Johns Hopkins University  
3400 N. Charles Street, Baltimore, MD 21218 U.S.A  
{hmei, jason}@cs.jhu.edu

## Abstract

Many events occur in the world. Some event types are stochastically excited or inhibited—in the sense of having their probabilities elevated or decreased—by patterns in the sequence of previous events. Discovering such patterns can help us predict *which type* of event will happen next and *when*. We model streams of discrete events in continuous time, by constructing a **neurally self-modulating multivariate point process** in which the intensities of multiple event types evolve according to a novel **continuous-time LSTM**. This generative model allows past events to influence the future in complex and realistic ways, by conditioning future event intensities on the hidden state of a recurrent neural network that has consumed the stream of past events. Our model has desirable qualitative properties. It achieves competitive likelihood and predictive accuracy on real and synthetic datasets, including under missing-data conditions.

## 1 Introduction

Some events in the world are correlated. A single event, or a pattern of events, may help to cause or prevent future events. We are interested in learning the distribution of sequences of events (and in future work, the causal structure of these sequences). The ability to discover correlations among events is crucial to accurately predict the future of a sequence given its past, i.e., which events are likely to happen next and when they will happen.

We specifically focus on sequences of discrete events in continuous time (“event streams”). Modeling such sequences seems natural and useful in many applied domains:

- *Medical events*. Each patient has a sequence of acute incidents, doctor’s visits, tests, diagnoses, and medications. By learning from previous patients what sequences tend to look like, we could predict a new patient’s future from their past.
- *Consumer behavior*. Each online consumer has a sequence of online interactions. By modeling the distribution of sequences, we can learn purchasing patterns. Buying cookies may temporarily depress purchases of all desserts, yet increase the probability of buying milk.
- *“Quantified self” data*. Some individuals use cellphone apps to record their behaviors—eating, traveling, working, sleeping, waking. By anticipating behaviors, an app could perform helpful supportive actions, including issuing reminders and placing advance orders.
- *Social media actions*. Previous posts, shares, comments, messages, and likes by a set of users are predictive of their future actions.
- Other event streams arise in *news*, *animal behavior*, *dialogue*, *music*, etc.

A basic model for event streams is the **Poisson process** (Palm, 1943), which assumes that events occur independently of one another. In a **non-homogenous Poisson process**, the (infinitesimal) probability of an event happening at time  $t$  may vary with  $t$ , but it is still independent of other events. A **Hawkes process** (Hawkes, 1971; Liniger, 2009) supposes that past events can temporarily *raise* the probability of future events, assuming that such excitation is ① positive, ② additive over the past events, and ③ exponentially decaying with time.



A final motivation is that one might wish to *intervene* in a medical, economic, or social event stream so as to improve the future course of events. Appendix D discusses our plans to deploy our model family as an environment model within reinforcement learning, where an agent controls some events.

## 2 Notation

We are interested in constructing distributions over **event streams**  $(k_1, t_1), (k_2, t_2), \dots$ , where each  $k_i \in \{1, 2, \dots, K\}$  is an event type and  $0 < t_1 < t_2 < \dots$  are times of occurrence.<sup>1</sup> That is, there are  $K$  types of events, tokens of which are observed to occur in continuous time.

For any distribution  $P$  in our proposed family, an event stream is almost surely infinite. However, when we observe the process only during a time interval  $[0, T]$ , the number  $I$  of observed events is almost surely finite. The *log-likelihood*  $\ell$  of the model  $P$  given these  $I$  observations is

$$\left( \sum_{i=1}^I \log P((k_i, t_i) \mid \mathcal{H}_i, \Delta t_i) \right) + \log P(t_{I+1} > T \mid \mathcal{H}_I) \quad (1)$$

where the **history**  $\mathcal{H}_i$  is the prefix sequence  $(k_1, t_1), (k_2, t_2), \dots, (k_{i-1}, t_{i-1})$ , and  $\Delta t_i \stackrel{\text{def}}{=} t_i - t_{i-1}$ , and  $P((k_i, t_i) \mid \mathcal{H}_i, \Delta t_i) dt$  is the probability that the *next* event occurs at time  $t_i$  and has type  $k_i$ .

Throughout the paper, the subscript  $i$  usually denotes quantities that affect the distribution of the next event  $(k_i, t_i)$ . These quantities depend only on the history  $\mathcal{H}_i$ .

We use (lowercase) Greek letters for parameters related to the classical Hawkes process, and Roman letters for other quantities, including hidden states and affine transformation parameters. We denote vectors by bold lowercase letters such as  $\mathbf{s}$  and  $\boldsymbol{\mu}$ , and matrices by bold capital Roman letters such as  $\mathbf{U}$ . Subscripted bold letters denote distinct vectors or matrices (e.g.,  $\mathbf{w}_k$ ). Scalar quantities, including vector and matrix elements such as  $s_k$  and  $\alpha_{j,k}$ , are written without bold. Capitalized scalars represent upper limits on lowercase scalars, e.g.,  $1 \leq k \leq K$ . Function symbols are notated like their return type. All  $\mathbb{R} \rightarrow \mathbb{R}$  functions are extended to apply elementwise to vectors and matrices.

## 3 The Model

In this section, we first review Hawkes processes, and then introduce our model one step at a time.

Formally, generative models of event streams are **multivariate point processes**. A (temporal) point process is a probability distribution over  $\{0, 1\}$ -valued functions on a given time interval (for us,  $[0, \infty)$ ). A multivariate point process is formally a distribution over  $K$ -tuples of such functions. The  $k^{\text{th}}$  function indicates the times at which events of type  $k$  occurred, by taking value 1 at those times.

### 3.1 Hawkes Process: A Self-Exciting Multivariate Point Process (SE-MPP)

A basic model of event streams is the **non-homogeneous multivariate Poisson process**. It assumes that an event of type  $k$  occurs at time  $t$ —more precisely, in the infinitesimally wide interval  $[t, t + dt)$ —with probability  $\lambda_k(t)dt$ . The value  $\lambda_k(t) \geq 0$  can be regarded as a rate per unit time, just like the parameter  $\lambda$  of an ordinary Poisson process.  $\lambda_k$  is known as the **intensity function**, and the total intensity of all event types is given by  $\lambda(t) = \sum_{k=1}^K \lambda_k(t)$ .

A well-known generalization that captures interactions is the **self-exciting multivariate point process (SE-MPP)**, or **Hawkes process** (Hawkes, 1971; Liniger, 2009), in which past events  $h$  from the history conspire to *raise* the intensity of each type of event. Such excitation is positive, additive over the past events, and exponentially decaying with time:

$$\lambda_k(t) = \mu_k + \sum_{h: t_h < t} \alpha_{h,k} \exp(-\delta_{h,k}(t - t_h)) \quad (2)$$

where  $\mu_k \geq 0$  is the base intensity of event type  $k$ ,  $\alpha_{j,k} \geq 0$  is the degree to which an event of type  $j$  initially excites type  $k$ , and  $\delta_{j,k} > 0$  is the decay rate of that excitation. When an event occurs, all intensities are elevated to various degrees, but then will decay toward their base rates  $\boldsymbol{\mu}$ .

<sup>1</sup>More generally, one could allow  $0 \leq t_1 \leq t_2 \leq \dots$ , where  $t_i$  is a **immediate event** if  $t_{i-1} = t_i$  and a **delayed event** if  $t_{i-1} < t_i$ . It is not too difficult to extend our model to assign positive probability to immediate events, but we will disallow them here for simplicity.

### 3.2 Self-Modulating Multivariate Point Processes

The positivity constraints in the Hawkes process limit its expressivity. First, the positive interaction parameters  $\alpha_{j,k}$  fail to capture inhibition effects, in which past events *reduce* the intensity of future events. Second, the positive base rates  $\mu$  fail to capture the inherent inertia of some events, which are unlikely until their cumulative excitation by past events crosses some threshold. To remove such limitations, we introduce two *self-modulating* models. Here the intensities of future events are stochastically *modulated* by the past history, where the term “modulation” is meant to encompass both excitation and inhibition. The intensity  $\lambda_k(t)$  can even fluctuate non-monotonically between successive events, because the competing excitatory and inhibitory influences may decay at different rates.

#### 3.2.1 Hawkes Process with Inhibition: A Decomposable Self-Modulating MPP (D-SM-MPP)

Our first move is to enrich the Hawkes model’s expressiveness while still maintaining its decomposable structure. We relax the positivity constraints on  $\alpha_{j,k}$  and  $\mu_k$ , allowing them to range over  $\mathbb{R}$ , which allows *inhibition* ( $\alpha_{j,k} < 0$ ) and *inertia* ( $\mu_k < 0$ ). However, the resulting total activation could now be negative. We therefore pass it through a non-linear **transfer function**  $f_k : \mathbb{R} \rightarrow \mathbb{R}_+$  to obtain a positive intensity function as required:

$$\lambda_k(t) = f_k(\tilde{\lambda}_k(t)) \quad (3a) \quad \tilde{\lambda}_k(t) = \mu_k + \sum_{h:t_h < t} \alpha_{k_h,k} \exp(-\delta_{k_h,k}(t - t_h)) \quad (3b)$$

As  $t$  increases between events, the intensity  $\lambda_k(t)$  may both rise and fall, but eventually approaches the base rate  $f(\mu_k + 0)$ , as the influence of each previous event still decays toward 0 at a rate  $\delta_{j,k} > 0$ .

What non-linear function  $f_k$  should we use? The ReLU function  $f(x) = \max(x, 0)$  is not strictly positive as required. A better choice is the scaled “softplus” function  $f(x) = s \log(1 + \exp(x/s))$ , which approaches ReLU as  $s \rightarrow 0$ . We learn a separate scale parameter  $s_k$  for each event type  $k$ , which adapts to the rate of that type. So we instantiate (3a) as  $\lambda_k(t) = f_k(\tilde{\lambda}_k(t)) = s_k \log(1 + \exp(\tilde{\lambda}_k(t)/s_k))$ . Appendix A.1 graphs this and motivates the “softness” and the scale parameter.

#### 3.2.2 Neural Hawkes Process: A Neurally Self-Modulating MPP (N-SM-MPP)

Our second move removes the restriction that the past events have independent, additive influence on  $\tilde{\lambda}_k(t)$ . Rather than predict  $\tilde{\lambda}_k(t)$  as a simple summation (3b), we now use a recurrent neural network. This allows learning a complex dependence of the intensities on the number, order, and timing of past events. We refer to our model as a **neural Hawkes process**.

Just as before, each event type  $k$  has an time-varying intensity  $\lambda_k(t)$ , which jumps discontinuously at each new event, and then drifts continuously toward a baseline intensity. In the new process, however, these dynamics are controlled by a hidden state vector  $\mathbf{h}(t) \in (-1, 1)^D$ , which in turn depends on a vector  $\mathbf{c}(t) \in \mathbb{R}^D$  of memory cells in a **continuous-time LSTM**.<sup>2</sup> This novel recurrent neural network architecture is inspired by the familiar discrete-time LSTM (Hochreiter and Schmidhuber, 1997; Graves, 2012). The difference is that in the continuous interval following an event, each memory cell  $c$  *exponentially decays* at some rate  $\delta$  toward some steady-state value  $\bar{c}$ .

At each time  $t > 0$ , we obtain the intensity  $\lambda_k(t)$  by (4a), where (4b) shows how the hidden states  $\mathbf{h}(t)$  are continually obtained from the memory cells  $\mathbf{c}(t)$  as the cells decay:

$$\lambda_k(t) = f_k(\mathbf{w}_k^\top \mathbf{h}(t)) \quad (4a) \quad \mathbf{h}(t) = \mathbf{o}_i \odot (2\sigma(2\mathbf{c}(t)) - 1) \text{ for } t \in (t_{i-1}, t_i] \quad (4b)$$

This says that on the interval  $(t_{i-1}, t_i]$ —in other words, after event  $i-1$  up until event  $i$  occurs at some time  $t_i$ —the  $\mathbf{h}(t)$  defined by equation (4b) determines the intensity functions via equation (4a). So for  $t$  in this interval, according to the model,  $\mathbf{h}(t)$  is a sufficient statistic of the history  $(\mathcal{H}_i, t - t_{i-1})$  with respect to future events (see equation (1)).  $\mathbf{h}(t)$  is analogous to  $\mathbf{h}_i$  in an LSTM language model (Mikolov et al., 2010), which summarizes the past event sequence  $k_1, \dots, k_{i-1}$ . But in our decay architecture, it will also reflect the interarrival times  $t_1 - 0, t_2 - t_1, \dots, t_{i-1} - t_{i-2}, t - t_{i-1}$ . This interval  $(t_{i-1}, t_i]$  ends when the next event  $k_i$  stochastically occurs at some time  $t_i$ . At this point, the continuous-time LSTM reads  $(k_i, t_i)$  and updates the current (decayed) hidden cells  $\mathbf{c}(t)$  to new initial values  $\mathbf{c}_{i+1}$ , based on the current (decayed) hidden state  $\mathbf{h}(t_i)$ .

<sup>2</sup>We use one-layer LSTMs with  $D$  hidden units in our present experiments, but a natural extension is to use multi-layer (“deep”) LSTMs (Graves et al., 2013), in which case  $\mathbf{h}(t)$  is the hidden state of the top layer.

How does the continuous-time LSTM make those updates? Other than depending on decayed values, the update formulas resemble the discrete-time case:<sup>3</sup>

$$\mathbf{i}_{i+1} \leftarrow \sigma(\mathbf{W}_i \mathbf{k}_i + \mathbf{U}_i \mathbf{h}(t_i) + \mathbf{d}_i) \quad (5a) \quad \mathbf{c}_{i+1} \leftarrow \mathbf{f}_{i+1} \odot \mathbf{c}(t_i) + \mathbf{i}_{i+1} \odot \mathbf{z}_{i+1} \quad (6a)$$

$$\mathbf{f}_{i+1} \leftarrow \sigma(\mathbf{W}_f \mathbf{k}_i + \mathbf{U}_f \mathbf{h}(t_i) + \mathbf{d}_f) \quad (5b) \quad \bar{\mathbf{c}}_{i+1} \leftarrow \bar{\mathbf{f}}_{i+1} \odot \bar{\mathbf{c}}_i + \bar{\mathbf{i}}_{i+1} \odot \mathbf{z}_{i+1} \quad (6b)$$

$$\mathbf{z}_{i+1} \leftarrow 2\sigma(\mathbf{W}_z \mathbf{k}_i + \mathbf{U}_z \mathbf{h}(t_i) + \mathbf{d}_z) - 1 \quad (5c) \quad \delta_{i+1} \leftarrow f(\mathbf{W}_d \mathbf{k}_i + \mathbf{U}_d \mathbf{h}(t_i) + \mathbf{d}_d) \quad (6c)$$

$$\mathbf{o}_{i+1} \leftarrow \sigma(\mathbf{W}_o \mathbf{k}_i + \mathbf{U}_o \mathbf{h}(t_i) + \mathbf{d}_o) \quad (5d)$$

The vector  $\mathbf{k}_i \in \{0, 1\}^K$  is the  $i^{\text{th}}$  input: a one-hot encoding of the new event  $k_i$ , with non-zero value only at the entry indexed by  $k_i$ . The above formulas will make a discrete update to the LSTM state. They resemble the discrete-time LSTM, but there are two differences. First, the updates do not depend on the “previous” hidden state from just after time  $t_{i-1}$ , but rather its value  $\mathbf{h}(t_i)$  at time  $t_i$ , after it has decayed for a period of  $t_i - t_{i-1}$ . Second, equations (6b)–(6c) are new. They define how in future, as  $t > t_i$  increases, the elements of  $\mathbf{c}(t)$  will continue to deterministically decay (at different rates) from  $\mathbf{c}_{i+1}$  toward targets  $\bar{\mathbf{c}}_{i+1}$ . Specifically,  $\mathbf{c}(t)$  is given by (7), which continues to control  $\mathbf{h}(t)$  and thus  $\lambda_k(t)$  (via (4), except that  $i$  has now increased by 1).

$$\mathbf{c}(t) \stackrel{\text{def}}{=} \bar{\mathbf{c}}_{i+1} + (\mathbf{c}_{i+1} - \bar{\mathbf{c}}_{i+1}) \exp(-\delta_{i+1}(t - t_i)) \text{ for } t \in (t_i, t_{i+1}] \quad (7)$$

In short, not only does (6a) define the usual cell values  $\mathbf{c}_{i+1}$ , but equation (7) defines  $\mathbf{c}(t)$  on  $\mathbb{R}_{>0}$ . On the interval  $(t_i, t_{i+1}]$ ,  $\mathbf{c}(t)$  follows an exponential curve that begins at  $\mathbf{c}_{i+1}$  (in the sense that  $\lim_{t \rightarrow t_i^+} \mathbf{c}(t) = \mathbf{c}_{i+1}$ ) and decays toward  $\bar{\mathbf{c}}_{i+1}$  (which it would approach as  $t \rightarrow \infty$ , if extrapolated).

A schematic example is shown in Figure 1. As in the previous models,  $\lambda_k(t)$  drifts deterministically between events toward some base rate. But the neural version is different in three ways: ① The base rate is not a constant  $\mu_k$ , but shifts upon each event.<sup>4</sup> ② The drift can be non-monotonic, because the excitatory and inhibitory influences on  $\lambda_k(t)$  from different elements of  $\mathbf{h}(t)$  may decay at different rates. ③ The sigmoidal transfer function means that the behavior of  $\mathbf{h}(t)$  itself is a little more interesting than exponential decay. Suppose that  $\mathbf{c}_i$  is very negative but increases toward a target  $\bar{\mathbf{c}}_i > 0$ . Then  $\mathbf{h}(t)$  will stay close to  $-1$  for a while and then will rapidly rise past 0. This usefully lets us model a delayed response (e.g. the last green segment in Figure 1).

We point out two behaviors that are naturally captured by our LSTM’s “forget” and “input” gates:

- if  $\mathbf{f}_{i+1} \approx 1$  and  $\mathbf{i}_{i+1} \approx 0$ , then  $\mathbf{c}_{i+1} \approx \mathbf{c}(t_i)$ . So  $\mathbf{c}(t)$  and  $\mathbf{h}(t)$  will be *continuous* at  $t_i$ . There is no jump due to event  $i$ , though the steady-state target may change.
- if  $\bar{\mathbf{f}}_{i+1} \approx 1$  and  $\bar{\mathbf{i}}_{i+1} \approx 0$ , then  $\bar{\mathbf{c}}_{i+1} \approx \bar{\mathbf{c}}_i$ . So although there may be a jump in activation, it is temporary. The memory cells will decay toward the same steady states as before.

Among other benefits, this lets us fit datasets in which (as is common) some pairs of event types do *not* influence one another. Appendix A.3 explains why all the models in this paper have this ability.

The drift of  $\mathbf{c}(t)$  between events controls how the system’s expectations about future events change as more time elapses with no event having yet occurred. Equation (7) chooses a moderately flexible parametric form for this drift function (see Appendix D for some alternatives). Equation (6a) was designed so that  $\mathbf{c}$  in an LSTM could learn to count past events with discrete-time exponential discounting; and (7) can be viewed as extending that to continuous-time exponential discounting.

Our memory cell vector  $\mathbf{c}(t)$  is a *deterministic* function of the past history  $(\mathcal{H}_i, t - t_i)$ .<sup>5</sup> Thus, the event intensities at any time are also deterministic via equation (4). The stochastic part of the model is the random choice—based on these intensities—of *which* event happens next and *when* it happens. The events are in competition: an event with high intensity is likely to happen sooner than an event with low intensity, and whichever one happens first is fed back into the LSTM. If no event type has high intensity, it may take a long time for the next event to occur.

Training the model means learning the LSTM parameters in equations (5) and (6c) along with the other parameters mentioned in this section, namely  $s_k \in \mathbb{R}$  and  $\mathbf{w}_k \in \mathbb{R}^D$  for  $k \in \{1, 2, \dots, K\}$ .

<sup>3</sup>The upright-font subscripts  $i, f, z$  and  $o$  are not variables, but constant labels that distinguish different  $\mathbf{W}$ ,  $\mathbf{U}$  and  $\mathbf{d}$  tensors. The  $\bar{\mathbf{f}}$  and  $\bar{\mathbf{i}}$  in equation (6b) are defined analogously to  $\mathbf{f}$  and  $\mathbf{i}$  but with different weights.

<sup>4</sup>Equations (4b) and (7) imply that after event  $i - 1$ , the base rate jumps to  $f_k(\mathbf{w}^\top (\mathbf{o}_i \odot (2\sigma(2\bar{\mathbf{c}}_i) - 1)))$ .

<sup>5</sup>Appendix A.2 explains how our LSTM handles the start and end of the sequence.

## 4 Algorithms

For the proposed models, the log-likelihood (1) of the parameters turns out to be given by a simple formula—the sum of the log-intensities of the events that happened, at the times they happened, minus an integral of the total intensities over the observation interval  $[0, T]$ :

$$\ell = \sum_{i:t_i \leq T} \log \lambda_{k_i}(t_i) - \underbrace{\int_{t=0}^T \lambda(t) dt}_{\text{call this } \Lambda} \quad (8)$$

The full derivation is given in Appendix B.1. Intuitively, the  $-\Lambda$  term (which is  $\leq 0$ ) sums the log-probabilities of infinitely many *non*-events. Why? The probability that there was *not* an event of any type in the infinitesimally wide interval  $[t, t + dt)$  is  $1 - \lambda(t)dt$ , whose log is  $-\lambda(t)dt$ .

We can locally maximize  $\ell$  using any stochastic gradient method. A detailed recipe is given in Appendix B.2, including the Monte Carlo trick we use to handle the integral in equation (8).

If we wish to draw random sequences from the model, we can adopt the thinning algorithm (Lewis and Shedler, 1979; Liniger, 2009) that is commonly used for the Hawkes process. See Appendix B.3.

Given an event stream prefix  $(k_1, t_1), (k_2, t_2), \dots, (k_{i-1}, t_{i-1})$ , we may wish to predict the *time* and *type* of the single next event. The next event’s time  $t_i$  has density  $p_i(t) = P(t_i = t \mid \mathcal{H}_i) = \lambda(t) \exp\left(-\int_{t_{i-1}}^t \lambda(s) ds\right)$ . To predict a single time whose expected  $L_2$  loss is as low as possible, we should choose  $\hat{t}_i = \mathbb{E}[t_i \mid \mathcal{H}_i] = \int_{t_{i-1}}^{\infty} t p_i(t) dt$ . Given the next event time  $t_i$ , the most likely type would be  $\arg\max_k \lambda_k(t_i)/\lambda(t_i)$ , but the most likely next event type *without* knowledge of  $t_i$  is  $\hat{k}_i = \arg\max_k \int_{t_{i-1}}^{\infty} \frac{\lambda_k(t)}{\lambda(t)} p_i(t) dt$ . The integrals in the preceding equations can be estimated by Monte Carlo sampling much as before (Appendix B.2). For event type prediction, we recommend a paired comparison that uses the same  $t$  values for each  $k$  in the  $\arg\max$ ; this also lets us share the  $\lambda(t)$  and  $p_i(t)$  computations across all  $k$ .

## 5 Related Work

The Hawkes process has been widely used to model event streams, including for topic modeling and clustering of text document streams (He et al., 2015; Du et al., 2015a), constructing and inferring network structure (Yang and Zha, 2013; Choi et al., 2015; Etesami et al., 2016), personalized recommendations based on users’ temporal behavior (Du et al., 2015b), discovering patterns in social interaction (Guo et al., 2015; Lukasik et al., 2016), learning causality (Xu et al., 2016), and so on.

Recent interest has focused on expanding the expressivity of Hawkes processes. Zhou et al. (2013) describe a self-exciting process that removes the assumption of exponentially decaying influence (as we do). They replace the scaled-exponential summands in equation (2) with learned positive functions of time (the choice of function again depends on  $k_i, k$ ). Lee et al. (2016) generalize the constant excitation parameters  $\alpha_{j,k}$  to be stochastic, which increases expressivity. Our model also allows non-constant interactions between event types, but arranges these via deterministic, instead of stochastic, functions of continuous-time LSTM hidden states. Wang et al. (2016) consider non-linear effects of past history on the future, by passing the intensity functions of the Hawkes process through a non-parametric isotonic link function  $g$ , which is in the same place as our non-linear function  $f_k$ . In contrast, our  $f_k$  has a fixed parametric form (learning only the scale parameter), and is approximately linear when  $x$  is large. This is because we model non-linearity (and other complications) with a continuous-time LSTM, and use  $f_k$  only to ensure positivity of the intensity functions.

Du et al. (2016) independently combined Hawkes processes with recurrent neural networks (and Xiao et al. (2017a) propose an advanced way of estimating the parameters of that model). However, Du et al.’s architecture is different in several respects. They use standard discrete-time LSTMs without our decay innovation, so they must encode the intervals between past events as explicit numerical inputs to the LSTM. They have only a single intensity function  $\lambda(t)$ , and it simply decays exponentially toward 0 between events, whereas our more modular model creates separate (potentially transferrable) functions  $\lambda_k(t)$ , each of which allows complex and non-monotonic dynamics en route to a non-zero steady state intensity. Some structural limitations of their design are that  $t_i$  and  $k_i$  are conditionally independent given  $\mathbf{h}$  (they are determined by separate distributions), and that their model cannot avoid a positive probability of extinction at all times. Finally, since they take

$f = \exp$ , the effect of their hidden units on intensity is effectively multiplicative, whereas we take  $f = \text{softplus}$  to get an approximately additive effect inspired by the classical Hawkes process. Our rationale is that additivity is useful to capture independent (disjunctive) causes; at the same time, the hidden units that our model adds up can each capture a complex joint (conjunctive) cause.

## 6 Experiments<sup>6</sup>

We fit our various models on several simulated and real-world datasets, and evaluated them in each case by the *log-probability* that they assigned to held-out data. We also compared our approach with that of Du et al. (2016) on their *prediction* task. The datasets that we use in this paper range from one extreme with only  $K = 2$  event types but mean sequence length  $> 2000$ , to the other extreme with  $K = 5000$  event types but mean sequence length 3. Dataset details can be found in Table 1 in Appendix C.1. Training details (e.g., hyperparameter selection) can be found in Appendix C.2.

### 6.1 Synthetic Datasets

In a pilot experiment with synthetic data (Appendix C.4), we confirmed that the neural Hawkes process generates data that is not well modeled by training an ordinary Hawkes process, but that ordinary Hawkes data can be successfully modeled by training an neural Hawkes process.

In this experiment, we were not limited to measuring the likelihood of the models on the stochastic event sequences. We also knew the true latent intensities of the generating process, so we were able to directly measure whether the trained models predicted these intensities accurately. The pattern of results was similar.

### 6.2 Real-World Media Datasets

**Retweets Dataset** (Zhao et al., 2015). On Twitter, *novel* tweets are generated from some distribution, which we do not model here. Each novel tweet serves as the beginning-of-stream event (see Appendix A.2) for a subsequent stream of *retweet* events. We model the dynamics of these streams: how retweets by various types of users ( $K = 3$ ) predict later retweets by various types of users.

Details of the dataset and its preparation are given in Appendix C.5. The dataset is interesting for its temporal pattern. People like to retweet an interesting post soon after it is created and retweeted by others, but may gradually lose interest, so the intervals between retweets become longer over time. In other words, the stream begins in a *self-exciting state*, in which previous retweets increase the intensities of future retweets, but eventually interest dies down and events are less able to excite one another. The decomposable models are essentially incapable of modeling such a phase transition, but our neural model should have the capacity to do so.

We generated learning curves (Figure 2) by training our models on increasingly long prefixes of the training set. As we can see, our self-modulating processes *significantly* outperform the Hawkes process at *all* training sizes. There is no obvious *a priori* reason to expect inhibition or even inertia in this application domain, which explains why the D-SM-MPP makes only a small improvement over the Hawkes process when the latter is well-trained. But D-SM-MPP requires much less data, and also has more stable behavior (smaller error bars) on small datasets. Our neural model is even better. Not only does it do better on the average stream, but its *consistent* superiority over the other two models is shown by the per-stream scatterplots in Figure 3, demonstrating the importance of our model’s neural component even with large datasets.

**MemeTrack Dataset** (Leskovec and Krevl, 2014). This dataset is similar in conception to Retweets, but with many more event types ( $K = 5000$ ). It considers the reuse of fixed phrases, or “memes,” in online media. It contains time-stamped instances of meme use in articles and posts from 1.5 million different blogs and news sites. We model how the future occurrence of a meme is affected by its past trajectory across different websites—that is, given one meme’s past trajectory across websites, when and where it will be mentioned again.

On this dataset,<sup>7</sup> the advantage of our full neural models was dramatic, yielding cross-entropy per event of around  $-8$  relative to the  $-15$  of D-SM-MPP—which in turn is *far* above the  $-800$  of the

<sup>6</sup>Our code and data are available at <https://github.com/HMEIatJHU/neurawkes>.

<sup>7</sup>Data preparation details are given in Appendix C.6.

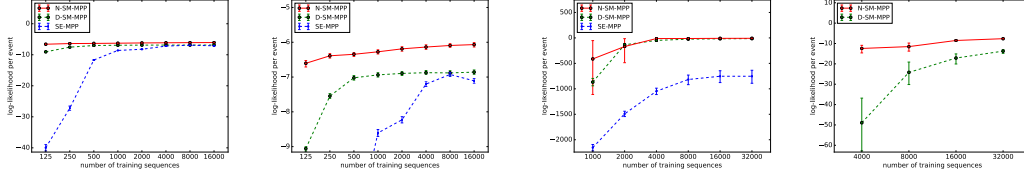


Figure 2: Learning curve (with 95% error bars) of all three models on the Retweets (left two) and MemeTrack (right two) datasets. Our neural model significantly outperforms our decomposable model (right graph of each pair), and both significantly outperform the Hawkes process (left of each pair—same graph zoomed out).

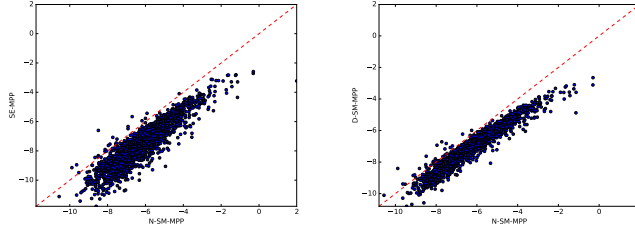


Figure 3: Scatterplots of N-SM-MPP vs. SE-MPP (left) and N-SM-MPP vs. D-SM-MPP (right), comparing the held-out log-likelihood of the two models (when trained on our full Retweets training set) with respect to *each* of the 2000 test sequences. Nearly all points fall to the right of  $y = x$ , since N-SM-MPP (the neural Hawkes process) is consistently more predictive than our non-neural model and the Hawkes process.

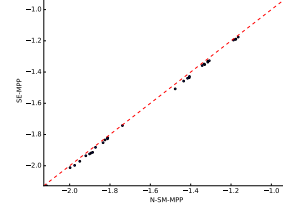


Figure 4: Scatterplot of N-SM-MPP vs. SE-MPP, comparing their log-likelihoods with respect to *each* of the 31 incomplete sequences' test sets. All 31 points fall to the right of  $y = x$ .

Hawkes process. Figure 2 illustrates the persistent gaps among the models. A scatterplot similar to Figure 3 is given in Figure 13 of Appendix C.6. We attribute the poor performance of the Hawkes process to its failure to capture the latent properties of memes, such as their topic, political stance, or interestingness. This is a form of missing data (section 1), as we now discuss.

As the table in Appendix C.1 indicates, most memes in MemeTrack are uninteresting and give rise to only a short sequence of mentions. Thus the base mention probability is low. An ideal analysis would recognize that if a specific meme has been mentioned several times already, it is *a posteriori* interesting and will probably be mentioned in future as well. The Hawkes process cannot distinguish the interesting memes from the others, except insofar as they appear on more influential websites. By contrast, our D-SM-MPP can partly capture this inferential pattern by using *negative* base rates  $\mu$  to create “inertia” (section 3.2.1). Indeed, all 5000 of its learned  $\mu_k$  parameters were negative, with values ranging from  $-10$  to  $-30$ , which numerically yields 0 intensity and is hard to excite.

An ideal analysis would also recognize that if a specific meme has appeared mainly on conservative websites, it is *a posteriori* conservative and unlikely to appear on liberal websites in the future. The D-SM-MPP, unlike the Hawkes process, can again partly capture this, by having conservative websites *inhibit* liberal ones. Indeed, 24% of its learned  $\alpha$  parameters were negative. (We re-emphasize that this inhibition is merely a predictive effect—probably not a direct causal mechanism.)

And our N-SM-MPP process is even more powerful. The LSTM state aims to learn sufficient statistics for predicting the future, so it can learn hidden dimensions (which fall in  $(-1, 1)$ ) that encode useful posterior beliefs in boolean properties of the meme such as interestingness, conservativeness, timeliness, etc. The LSTM’s “long short-term memory” architecture explicitly allows these beliefs to persist indefinitely through time in the absence of new evidence, without having to be refreshed by redundant new events as in the decomposable models. Also, the LSTM’s hidden dimensions are computed by sigmoidal activation rather than softplus activation, and so can be used implicitly to perform logistic regression. The flat left side of the sigmoid resembles softplus and can model *inertia* as we saw above: it takes several mentions to establish interestingness. Symmetrically, the flat right side can model *saturation*: once the posterior probability of interestingness is at 80%, it cannot climb much farther no matter how many more mentions are observed.

A final potential advantage of the LSTM is that in this large- $K$  setting, it has fewer parameters than the other models (Appendix C.3), sharing statistical strength across event types (websites) to generalize better. The learning curves in Figure 2 suggest that on small data, the decomposable



(non-neural) models may overfit their  $O(K^2)$  interaction parameters  $\alpha_{j,k}$ . Our neural model only has to learn  $O(D^2)$  pairwise interactions among its  $D$  hidden nodes (where  $D \ll K$ ), as well as  $O(KD)$  interactions between the hidden nodes and the  $K$  event types. In this case,  $K = 5000$  but  $D = 64$ . This reduction by using latent hidden nodes is analogous to nonlinear latent factor analysis.

### 6.3 Modeling Streams With Missing Data

We set up an artificial experiment to more directly investigate the missing-data setting of section 1, where we do not observe *all* events during  $[0, T]$ , but train and test our model just as if we had.

We sampled synthetic event sequences from a standard Hawkes process (just as in our pilot experiment from 6.1), removed all the events of selected types, and then compared the neural Hawkes process (N-SM-MPP) with the Hawkes process (SE-MPP) as models of these censored sequences. Since we took  $K = 5$ , there were  $2^5 - 1 = 31$  ways to construct a dataset of censored sequences. As shown in Figure 4, for *each* of the 31 resulting datasets, training a neural Hawkes model achieves better generalization. Appendix A.4 discusses why this kind of behavior is to be expected.

### 6.4 Prediction Tasks—Medical, Social and Financial

To compare with Du et al. (2016), we evaluate our model on the *prediction* tasks and datasets that they proposed. The Financial Transaction dataset contains long streams of high frequency stock transactions for a single stock, with the two event types “buy” and “sell.” The electrical medical records (MIMIC-II) dataset is a collection of de-identified clinical visit records of Intensive Care Unit patients for 7 years. Each patient has a sequence of hospital visit events, and each event records its time stamp and disease diagnosis. The Stack Overflow dataset represents two years of user awards on a question-answering website: each user received a sequence of badges (of 22 different types).

We follow Du et al. (2016) and attempt to predict every held-out event  $(k_i, t_i)$  from its history  $\mathcal{H}_i$ , evaluating the prediction  $\hat{k}_i$  with 0-1 loss (yielding an error rate, or ER) and evaluating the prediction  $\hat{t}_i$  with L2 loss (yielding a root-mean-squared error, or RMSE). We make minimum Bayes risk predictions as explained in section 4. Figure 8 in Appendix C.7 shows that our model consistently outperforms that of Du et al. (2016) on event type prediction on all the datasets, although for time prediction neither model is consistently better.

### 6.5 Sensitivity to Number of Parameters

Does our method do well because of its flexible nonlinearities or just because it has more parameters? The answer is both. We experimented on the Retweets data with reducing the number of hidden units  $D$ . Our N-SM-MPP substantially outperformed SE-MPP (the Hawkes process) on held-out data even with very few parameters, although more parameters does even better:

number of hidden units	Hawkes	1	2	4	8	16	32	256
number of parameters	21	31	87	283	1011	3811	14787	921091
log-likelihood	-7.19	-6.51	-6.41	-6.36	-6.24	-6.18	-6.16	-6.10

We also tried halving  $D$  across several datasets, which had negligible effect, always decreasing held-out log-likelihood by  $< 0.2\%$  relative.

More information about model sizes is given in Appendix C.3. Note that the neural Hawkes process does not *always* have more parameters. When  $K$  is large, we can greatly reduce the number of params below that of a Hawkes process, by choosing  $D \ll K$ , as for MemeTrack in section 6.2.

## 7 Conclusion

We presented two extensions to the multivariate Hawkes process, a popular generative model of streams of typed, timestamped events. Past events may now either excite *or* inhibit future events. They do so by *sequentially* updating the state of a novel *continuous-time* recurrent neural network (LSTM). Whereas Hawkes sums the time-decaying influences of past events, we instead sum the time-decaying influences of the LSTM nodes. Our extensions to Hawkes aim to address real-world phenomena, missing data, and causal modeling. Empirically, we have shown that both extensions yield a significantly improved ability to predict the course of future events. There are several exciting avenues for further improvements (discussed in Appendix D), including embedding our model within a reinforcement learner to discover causal structure and learn an intervention policy.

## Acknowledgments

We are grateful to Facebook for enabling this work through a gift to the second author. Nan Du kindly helped us by making his code public and answering questions, and the NVIDIA Corporation kindly donated two Titan X Pascal GPUs. We also thank our lab group at Johns Hopkins University’s Center for Language and Speech Processing for helpful comments. The first version of this work appeared on arXiv in December 2016.

## References

- Ciprian Chelba, Tomas Mikolov, Mike Schuster, Qi Ge, Thorsten Brants, Phillipp Koehn, and Tony Robinson. One billion word benchmark for measuring progress in statistical language modeling. *Computing Research Repository*, arXiv:1312.3005, 2013. URL <http://arxiv.org/abs/1312.3005>.
- Edward Choi, Nan Du, Robert Chen, Le Song, and Jimeng Sun. Constructing disease network and temporal progression model via context-sensitive Hawkes process. In *Data Mining (ICDM), 2015 IEEE International Conference on*, pages 721–726. IEEE, 2015.
- Nan Du, Mehrdad Farajtabar, Amr Ahmed, Alexander J Smola, and Le Song. Dirichlet-Hawkes processes with applications to clustering continuous-time document streams. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 219–228. ACM, 2015a.
- Nan Du, Yichen Wang, Niao He, Jimeng Sun, and Le Song. Time-sensitive recommendation from recurrent user activities. In *Advances in Neural Information Processing Systems (NIPS)*, pages 3492–3500, 2015b.
- Nan Du, Hanjun Dai, Rakshit Trivedi, Utkarsh Upadhyay, Manuel Gomez-Rodriguez, and Le Song. Recurrent marked temporal point processes: Embedding event history to vector. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1555–1564. ACM, 2016.
- Jalal Etesami, Negar Kiyavash, Kun Zhang, and Kushagra Singhal. Learning network of multivariate Hawkes processes: A time series approach. *arXiv preprint arXiv:1603.04319*, 2016.
- Manuel Gomez Rodriguez, Jure Leskovec, and Bernhard Schölkopf. Structure and dynamics of information pathways in online media. In *Proceedings of the Sixth ACM International Conference on Web Search and Data Mining*, pages 23–32. ACM, 2013.
- Alex Graves. *Supervised Sequence Labelling with Recurrent Neural Networks*. Springer, 2012. URL <http://www.cs.toronto.edu/~graves/preprint.pdf>.
- Alex Graves, Navdeep Jaitly, and Abdel-rahman Mohamed. Hybrid speech recognition with deep bidirectional LSTM. In *Automatic Speech Recognition and Understanding (ASRU), 2013 IEEE Workshop on*, pages 273–278. IEEE, 2013.
- Fangjian Guo, Charles Blundell, Hanna Wallach, and Katherine Heller. The Bayesian echo chamber: Modeling social influence via linguistic accommodation. In *Proceedings of the Eighteenth International Conference on Artificial Intelligence and Statistics*, pages 315–323, 2015.
- Alan G Hawkes. Spectra of some self-exciting and mutually exciting point processes. *Biometrika*, 58(1):83–90, 1971.
- Xinran He, Theodoros Rekatsinas, James Foulds, Lise Getoor, and Yan Liu. Hawkestopic: A joint model for network inference and topic modeling from text-based cascades. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 871–880, 2015.
- Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8): 1735–1780, 1997.
- Andrej Karpathy, Justin Johnson, and Li Fei-Fei. Visualizing and understanding recurrent networks. *arXiv preprint arXiv:1506.02078*, 2015.

- Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2015.
- Young Lee, Kar Wai Lim, and Cheng Soon Ong. Hawkes processes with stochastic excitations. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2016.
- Jure Leskovec and Andrej Krevl. SNAP Datasets: Stanford large network dataset collection. <http://snap.stanford.edu/data>, June 2014.
- Peter A Lewis and Gerald S Shedler. Simulation of nonhomogeneous Poisson processes by thinning. *Naval Research Logistics Quarterly*, 26(3):403–413, 1979.
- Thomas Josef Liniger. *Multivariate Hawkes processes*. Diss., Eidgenössische Technische Hochschule ETH Zürich, Nr. 18403, 2009, 2009.
- Michal Lukasik, PK Srijith, Duy Vu, Kalina Bontcheva, Arkaitz Zubiaga, and Trevor Cohn. Hawkes processes for continuous time sequence classification: An application to rumour stance classification in Twitter. In *Proceedings of 54th Annual Meeting of the Association for Computational Linguistics*, pages 393–398, 2016.
- Tomas Mikolov, Martin Karafiát, Lukás Burget, Jan Cernocký, and Sanjeev Khudanpur. Recurrent neural network based language model. In *INTERSPEECH 2010, 11th Annual Conference of the International Speech Communication Association, Makuhari, Chiba, Japan, September 26-30, 2010*, pages 1045–1048, 2010.
- C. Palm. *Intensitätsschwankungen im Fernspreckverkehr*. Ericsson technics, no. 44. L. M. Ericsson, 1943. URL <https://books.google.com/books?id=5cy2NQAACAAJ>.
- Judea Pearl. Causal inference in statistics: An overview. *Statistics Surveys*, 3:96–146, 2009.
- Martin Sundermeyer, Hermann Ney, and Ralf Schluter. LSTM neural networks for language modeling. *Proceedings of INTERSPEECH*, 2012.
- Yichen Wang, Bo Xie, Nan Du, and Le Song. Isotonic Hawkes processes. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2016.
- Shuai Xiao, Mehrdad Farajtabar, Xiaojing Ye, Junchi Yan, Xiaokang Yang, Le Song, and Hongyuan Zha. Wasserstein learning of deep generative point process models. In *Advances in Neural Information Processing Systems 30*, 2017a.
- Shuai Xiao, Junchi Yan, Mehrdad Farajtabar, Le Song, Xiaokang Yang, and Hongyuan Zha. Joint modeling of event sequence and time series with attentional twin recurrent neural networks. *arXiv preprint arXiv:1703.08524*, 2017b.
- Hongteng Xu, Mehrdad Farajtabar, and Hongyuan Zha. Learning Granger causality for Hawkes processes. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2016.
- Shuang-hong Yang and Hongyuan Zha. Mixture of mutually exciting processes for viral diffusion. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 1–9, 2013.
- Omar F. Zaidan and Jason Eisner. Modeling annotators: A generative approach to learning from annotator rationales. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 31–40, 2008.
- Qingyuan Zhao, Murat A Erdogdu, Hera Y He, Anand Rajaraman, and Jure Leskovec. Seismic: A self-exciting point process model for predicting tweet popularity. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1513–1522. ACM, 2015.
- Ke Zhou, Hongyuan Zha, and Le Song. Learning triggering kernels for multi-dimensional Hawkes processes. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 1301–1309, 2013.

# Appendices

## A Model Details

In this appendix, we discuss some qualitative properties of our models and give details about how we handle boundary conditions.

### A.1 Discussion of the Transfer Function

As explained in section 3.2, when we allow *inhibition* and *inertia*, we need to pass the total activation through a non-linear **transfer function**  $f : \mathbb{R} \rightarrow \mathbb{R}_+$  to obtain a positive intensity function. This was our equation (3a), namely  $\lambda_k(t) = f(\tilde{\lambda}_k(t))$ .

What non-linear function  $f$  should we use? The ReLU function  $f(x) = \max(x, 0)$  seems at first a natural choice. However, it returns 0 for negative  $x$ ; we need to keep our intensities strictly positive at all times when an event could possibly occur, to avoid infinitely bad log-likelihood at training time or infinite log-loss at test time.

A better choice would be the “softplus” function  $f(x) = \log(1 + \exp(x))$ , which is strictly positive and approaches ReLU when  $x$  is far from 0. Unfortunately, “far from 0” is defined in units of  $x$ , so this choice would make our model sensitive to the units used to measure time. For example, if we switch the units of  $t$  from seconds to milliseconds, then the base intensity  $f(\mu_k)$  must become 1000 times lower, forcing  $\mu_k$  to be very negative and thus creating a much stronger inertial effect.

To avoid this problem, we introduce a scale parameter  $s > 0$  and define  $f(x) = s \log(1 + \exp(x/s))$ . The scale parameter  $s$  controls the curvature of  $f(x)$ , which approaches ReLU as  $s \rightarrow 0$ , as shown in Figure 5. We can regard  $f(x)$ ,  $x$ , and  $s$  as rates, with units of inverse time, so that  $f(x)/s$  and  $x/s$  are unitless quantities related by softplus. We actually learn a separate scale parameter  $s_k$  for each event type  $k$ , which will adapt to the rate of events of that type.

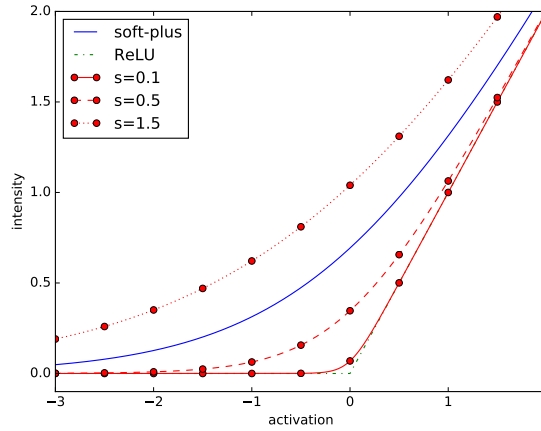


Figure 5: The softplus function is a soft approximation to a rectified linear unit (ReLU), approaching it as  $x$  moves away from 0. We use it to ensure a strictly positive intensity function. We incorporate a scale parameter  $s$  that controls the curvature.

### A.2 Boundary Conditions for the LSTM

We initialize the continuous-time LSTM’s hidden state to  $\mathbf{h}(0) = \mathbf{0}$ , and then have it read a special beginning-of-stream (BOS) event  $(k_0, t_0)$ , where  $k_0$  is a special event type (i.e., expanding the

LSTM’s input dimensionality by one) and  $t_0$  is set to be 0. Then equations (5)–(6) define  $\mathbf{c}_1$  (from  $\mathbf{c}_0 \stackrel{\text{def}}{=} \mathbf{0}$ ),  $\bar{\mathbf{c}}_1$ ,  $\delta_1$ , and  $\mathbf{o}_1$ . This is the initial configuration of the system as it waits for the first event to happen: this initial configuration determines the hidden state  $\mathbf{h}(t)$  and the intensity functions  $\lambda_k(t)$  over  $t \in (0, t_1]$

We do not generate the BOS event but only condition on it, which is why the log-likelihood formula (section 2) only sums over  $i = 1, 2, \dots$ . This design is well-suited to various settings. In some settings, time 0 is special. For example, if we release children into a carnival and observe the stream of their actions there, then BOS is the release event and no other events can possibly precede it. In other settings, data before time 0 are simply missing, e.g., the observation of a patient starts in midlife; nonetheless, BOS in this case usefully indicates the beginning of the *observed* sequence. In both kinds of settings, the initial configuration just after reading BOS characterizes the model’s belief about the unknown state of the true system just after time 0, as it waits for event 1. Computing the initial configuration by explicitly transitioning on BOS ensures that the initial hidden state  $\mathbf{h}(0^+) \stackrel{\text{def}}{=} \lim_{t \rightarrow 0^+} \mathbf{h}(t)$  falls in the space of hidden states achievable by LSTM transitions. More important, in future work, we will be able to attach metadata about the sequence as a “mark” to the BOS event (see footnote 13), and the LSTM can learn how these metadata affect the initial configuration.

To allow finite streams, we could optionally choose to identify one of the observable types in  $\{1, 2, \dots, K\}$  as a special end-of-stream (EOS) event after which the stream cannot possibly continue. If the model generates EOS, all intensities are permanently forced to 0—the LSTM is no longer consulted, so it is not necessary for the model parameters to explain why no further events are observed on the interval  $[0, T]$ : that is, the second term of equation (1) can be omitted. The integral in equation (8) should therefore be taken from  $t = 0$  to the time of the EOS event or  $T$ , whichever is smaller.

### A.3 Closure Under Superposition

Decomposable models have the nice property that they are closed under superposition of event streams. Let  $\mathcal{E}$  and  $\mathcal{E}'$  be random event streams, on a common time interval  $[0, T]$  but over disjoint sets of event types. If each stream is distributed according to a Hawkes process, then their superposition—that is,  $\mathcal{E} \cup \mathcal{E}'$  sorted into temporally increasing order—is also distributed according to a Hawkes process. It is easy to exhibit parameters for such a process, using a block-diagonal matrix of  $\alpha_{j,k}$  so that the two sets of event types do not influence each other. The closure property also holds for our decomposable self-modulating process, and for the same simple reason.

This is important since in various real settings, some event types tend not to interact. For example, the activities of two people Jay and Kay rarely influence each other,<sup>8</sup> although they are simultaneously monitored and thus form a single observed stream of events. We want our model to handle such situations naturally, rather than insisting that Kay always reacts to what Jay does.

Thus, as section 3.2.2 noted, we have designed our neurally self-modulating process to preserve this ability to insulate event  $k$  from event  $j$ . By setting specific elements of  $\mathbf{w}_k$  to 0, one could ensure that the intensity function  $\lambda_k(t)$  depends on only a subset  $S$  of the LSTM hidden nodes. Then by setting specific LSTM parameters, one would make the nodes in  $S$  insensitive to events of type  $j$ : events of type  $j$  should open these nodes’ forget gates ( $\mathbf{f} = 1$ ) and close their input gates ( $\mathbf{i} = 0$ )—as section 3.2.2 suggested—so that their cell memories  $\mathbf{c}(t)$  and hidden states  $\mathbf{h}(t)$  do not change at all but continue decaying toward their previous steady-state values.<sup>9</sup> Now events of type  $j$  cannot affect the intensity  $\lambda_k(t)$ .

For example, the hidden states in  $S$  are affected in the same way when the LSTM reads  $(k, 1), (j, 3), (j, 8), (k, 12)$  as when it reads  $(k, 1), (k, 12)$ , even though the intervals  $\Delta t$  between successive events are different. In other words, the architecture “knows” that  $2 + 5 + 4 = 11$ . The simplicity of this solution is a consequence of how our design does not encode the time intervals

<sup>8</sup>Their surnames might be Box and Cox, after the 19th-century farce about a day worker and a night worker unknowingly renting the same room. But any pair of strangers would do.

<sup>9</sup>To be precise, we can achieve this arbitrarily closely, but not exactly, because a standard LSTM gate cannot be fully opened or closed. The openness is traditionally given by a sigmoid function and so falls in  $(0, 1)$ , never achieving 1 or 0 exactly unless we are willing to set parameters to  $\pm\infty$ . In practice this should not be an issue because relatively small weights can drive the sigmoid function extremely close to 1 and 0—in fact,  $\sigma(37) = 1$  in 64-bit floating-point arithmetic.

numerically, but only reacts to these intervals indirectly, through the interaction between the timing of events and the spontaneous decay of the hidden states. The memory cells of  $S$  decay for a total duration of 11 between the two  $k$  events, even if that interval has been divided into subintervals  $2 + 5 + 4$ .

With this method, we can explicitly construct a superposition process with LSTM state space  $\mathbb{R}^{d+d'}$ —the cross product of the state spaces  $\mathbb{R}^d$  and  $\mathbb{R}^{d'}$  of the original processes—in which Kay’s events are not influenced at all by Jay’s.

If we know *a priori* that particular event types interact only weakly, we can impose an appropriate prior on the neural Hawkes parameters. And in future work with large  $K$ , we plan to investigate the use of sparsity-inducing regularizers during parameter estimation, to create an inductive bias toward models that have limited interactions, without specifying which particular interactions are present.

Superposition is a formally natural operation on event streams. It barely arises for ordinary sequence models, such as language models, since the superposition of two sentences is not well-defined unless all of the words carry distinct real-valued timestamps. However, there is an analogue from formal language theory. The “shuffle” of two sentences is defined to be the set of *possible* interleavings of their words—i.e., the set of superpositions that could result from assigning increasing timestamps to the words of each sentence, without duplicates. It is a standard exercise to show that regular languages are closed under shuffle. This is akin to our remark that neural-Hawkes-distributed random variables are closed under superposition, and indeed uses a similar cross-product construction on the finite-state automata. An important difference is that the shuffle construction does not require disjoint alphabets in the way that ours requires disjoint sets of event types. This is because finite-state automata allow nondeterministic state transitions and our processes do not.

#### A.4 Missing Data Discussion

We discussed the case of missing data in section 1. Suppose the true complete-data distribution  $p^*$  is itself an unknown neural Hawkes process. As section 1 pointed out, a sufficient statistic for prediction from the incompletely observed past would be the posterior distribution over the true hidden neural state  $\mathbf{t}$  of the unknown process, which was reached by reading the *complete* past. We would ideally obtain our predictions by correctly modeling the missing observations and integrating over them. However, inference would be computationally quite expensive even if  $p^*$  were known, to say nothing of the case where  $p^*$  is unknown and we must integrate over its parameters as well.

We instead train a neural model that attempts to bypass these problems. The hope is that our model’s hidden state, after it reads only the observed *incomplete* past, will be nearly as predictive as the posterior distribution above.

We can illustrate the goal with reference to the experiment in section 6.3. There, the true complete-data distribution  $p^*$  happened to be a classical Hawkes process, but we censored some event types. We then modeled the observed incomplete sequence as if it were a complete sequence. In this setting, a Hawkes process will in general be unable to fit the data well, which is why the neural Hawkes process has an advantage in all 31 experiments.

What goes wrong with using the Hawkes model? Suppose that in the true Hawkes model  $p^*$ , type 1 is rare but strongly excites type 2 and type 3, which do not excite themselves or each other. Type 1 events are missing in the observed sequence.

What is the correct predictive distribution in this situation (with knowledge of  $p^*$ )? Seeing lots of type 2 events in a row suggests that they were preceded by a (single) missing type 1 event, which predicts a higher intensity for type 3 in future. The more type 2 events we see, the surer we are that there was a type 1 event, but we doubt that there were multiple type 1 events, so the predicted intensity of type 3 is expected to increase sublinearly as  $P(\text{type} = 1)$  approaches 1.

As neural networks are universal function approximators, a neural Hawkes model may be able to recognize and fit this sublinear behavior in the incomplete training data. However, if we fit only a Hawkes model to the incomplete training data, it would have to posit that type 2 excites type 3 directly, so the predicted intensity of type 3 would incorrectly increase linearly with the number of type 2 events.

## B Algorithmic Details

In this appendix, we elaborate on the details of algorithms.

### B.1 Likelihood Function

For the proposed models, given complete observations of an event stream over the time interval  $[0, T]$ , the log-likelihood of the parameters turns out to be given by the simple formula shown in section 4. We start by giving the full derivation of that formula, repeated here:

$$\ell = \sum_{i:t_i \leq T} \log \lambda_{k_i}(t_i) - \underbrace{\int_{t=0}^T \lambda(t) dt}_{\text{call this } \Lambda} \quad (8)$$

First, we define  $N(t) = |\{h : t_h \leq t\}|$  to be the count of events (of any type) preceding time  $t$ . So given the past history  $\mathcal{H}_i$ , the number of events in  $(t_{i-1}, t]$  is denoted as  $\Delta N(t_{i-1}, t) \stackrel{\text{def}}{=} N(t) - N(t_{i-1})$ . Let  $T_i > t_{i-1}$  be the random variable of the next event time and let  $K_{i+1}$  be the random variable of the next event type. The cumulative distribution function and probability density function of  $T_i$  (conditioned on  $\mathcal{H}_i$ ) are given by:

$$F(t) = P(T_i \leq t) = 1 - P(T_i > t) \quad (9a)$$

$$= 1 - P(\Delta N(t_{i-1}, t) = 0) \quad (9b)$$

$$= 1 - \exp\left(-\int_{t_{i-1}}^t \lambda(s) ds\right) \quad (9c)$$

$$= 1 - \exp(\Lambda(t_{i-1}) - \Lambda(t)) \quad (9d)$$

$$f(t) = \exp(\Lambda(t_{i-1}) - \Lambda(t)) \lambda(t) \quad (9e)$$

where  $\Lambda(t) = \int_0^t \lambda(s) ds$  and  $\lambda(t) = \sum_{k=1}^K \lambda_k(t)$ .

Moreover, given the past history  $\mathcal{H}_i$  and the next event time  $t_i$ , the distribution of  $k_i$  is given by:

$$P(K_i = k_i | t_i) = \frac{\lambda_{k_i}(t_i)}{\lambda(t_i)} \quad (10)$$

Therefore, we can derive the likelihood function as follows:

$$\mathcal{L} = \prod_{i:t_i \leq T} \mathcal{L}_i = \prod_{t_i \leq T} \{f(t_i) P(K_i = k_i | t_i)\} \quad (11a)$$

$$= \prod_{i:t_i \leq T} \{\exp(\Lambda(t_{i-1}) - \Lambda(t_i)) \lambda_{k_i}(t_i)\} \quad (11b)$$

and

$$\ell \stackrel{\text{def}}{=} \log \mathcal{L} \quad (12a)$$

$$= \sum_{i:t_i \leq T} \log \lambda_{k_i}(t_i) - \sum_{i:t_i \leq T} (\Lambda(t_i) - \Lambda(t_{i-1})) \quad (12b)$$

$$= \sum_{i:t_i \leq T} \log \lambda_{k_i}(t_i) - \Lambda(T) \quad (12c)$$

$$= \sum_{i:t_i \leq T} \log \lambda_{k_i}(t_i) - \int_{t=0}^T \lambda(t) dt \quad (12d)$$

### B.2 Monte Carlo Gradient and Training Speed

We can locally maximize the log-likelihood  $\ell$  from equation (8) using any stochastic gradient method. For this, we need to be able to get an unbiased estimate of the gradient  $\nabla \ell$  with respect to the model parameters. This is straightforward to obtain by back-propagation. The trick

---

**Algorithm 1** Integral Estimation (Monte Carlo)

---

**Input:** interval  $[0, T]$ ; model parameters and events  $(k_1, t_1), \dots$  for determining  $\lambda_j(t)$   
 $\Lambda \leftarrow 0; \nabla \Lambda \leftarrow \mathbf{0}$   
**for**  $N$  samples :  $\triangleright$  e.g., take  $N > 0$  proportional to  $T$   
  draw  $t \sim \text{Unif}(0, T)$   
  **for**  $j \leftarrow 1$  **to**  $K$  :  
     $\Lambda += \lambda_j(t)$   $\triangleright$  via current model parameters  
     $\nabla \Lambda += \nabla \lambda_j(t)$   $\triangleright$  via back-propagation  
 $\Lambda \leftarrow T\Lambda/N; \nabla \Lambda \leftarrow T\nabla \Lambda/N$   $\triangleright$  weight the samples  
**return**  $(\Lambda, \nabla \Lambda)$

---

for handling the integral in equation (8) is that the single function evaluation  $T\lambda(t)$  at a random  $t \sim \text{Unif}(0, T)$  gives an unbiased estimate of the entire integral—that is, its expected value is  $\Lambda$ . Its gradient via back-propagation is therefore an unbiased estimate of  $\nabla \Lambda$  (since gradient commutes with expectation). The Monte Carlo algorithm in Algorithm 1 averages over several samples to reduce the variance of this noisy estimator.

Each step of Adam training computes the gradient on a training sequence. With  $P$  params, this takes time  $O(IP)$  for Hawkes and  $O((I+M)P)$  for neural Hawkes, if  $I$  is the number of observed events and  $M$  is the number of samples used to estimate the integral. We take  $M = O(I)$  in practice (see Appendix C.2), so we have runtime  $O(IP)$  like Hawkes.

Note that our stochastic gradient is unbiased for any  $M$ ; large  $M$  merely reduces its variance. The gradient for the Hawkes process has 0 variance, since it has analytical form and does not require sampling at all.

### B.3 Thinning Algorithm for Sampling Sequences

If we wish to draw sequences from the self-modulating models of 3.2, we can adopt the thinning algorithm (Lewis and Shedler, 1979; Liniger, 2009) that is commonly used for the multivariate Hawkes process, as shown in Algorithm 2. We explain the algorithm here and illustrate its conception in Figure 6.

Suppose we have already sampled the first  $i - 1$  events. The  $K$  event types are now in a race to see who generates the next event. (Typically, the winning type will have relatively high intensity.) In our model, that next event will join the multivariate event stream as  $(k_i, t_i)$ , whereupon it updates the LSTM state and thus modulates the subsequent intensities that will be used to sample event  $i + 1$ .

How do we conduct the race? For each event type  $k$ , let the function  $\lambda_k^i : (t_{i-1}, \infty) \rightarrow \mathbb{R}_{\geq 0}$  map each time  $t$  to the intensity  $\lambda_k^i(t)$  that our model will define at time  $t$  provided that event  $i$  has not yet happened in the interval  $(t_{i-1}, t)$ . For each  $k$  independently, we draw the time  $t_{i,k}$  of the next event from the non-homogeneous Poisson process over  $(t_{i-1}, \infty)$  whose intensity function is  $\lambda_k^i$ . We then take  $t_i = \min_k t_{i,k}$  and  $k_i = \text{argmin}_k t_{i,k}$ . That is, we keep just the earliest of the  $K$  events. We cannot keep the rest because they are not correctly distributed according to the new intensities as updated by the earliest event.

But how do we draw the next event time  $t_{i,k}$  from the non-homogeneous Poisson process given by  $\lambda_k^i$ ? Recall from 3.1 that a draw from such a point process is actually a whole *set* of times in  $(t_{i-1}, \infty)$ : we will take  $t_{i,k}$  to be the earliest of these. In theory, this set is drawn by *independently* choosing at each time  $t \in (t_{i-1}, \infty)$ , with infinitesimal probability proportional to  $\lambda_k^i(t)$ , whether an event occurs. One could do this by *independently* applying rejection sampling at each time  $t$ : choose with larger probability  $\lambda^*$  whether a “proposed event” occurs at time  $t$ , and if it does, accept the proposed event with probability only  $\lambda_k^i(t)/\lambda^* \leq 1$ . This is equivalent to simultaneously drawing a set of proposed times from a *homogenous* Poisson process with constant rate  $\lambda^*$ , and then “thinning” that proposed set, as illustrated in Figure 6. This approach helps because it is easy to draw from the homogenous process: the intervals between successive proposed events are IID  $\text{Exp}(\lambda^*)$ , so it is easy to sample the events in sequence. The inner **repeat** loop in Algorithm 2 lazily carries out just enough of this infinite homogenous draw from  $\lambda^*$  to determine the time  $t_{i,k}$  of the earliest *accepted* event, which is the earliest event in the non-homogeneous draw from  $\lambda_k^i$ , as desired.



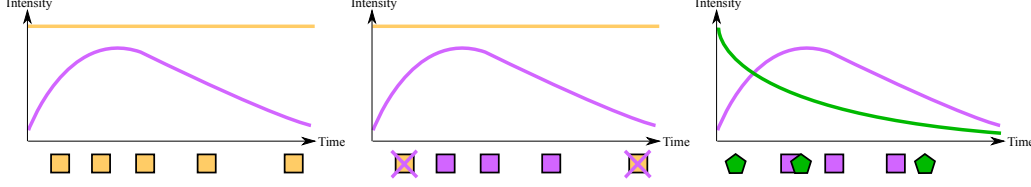


Figure 6: Sampling the next event, using the same visual notation as in Figure 1. The  $x$  axis shows a prefix of the infinite interval  $(t_{i-1}, \infty)$ . In the first graph, *gold* events are proposed from a homogeneous Poisson process with intensity  $\lambda^*$  (gold straight line). In the second graph, the purple curve  $\lambda_1^i$  randomly accepts some of these gold events, with probability  $\lambda_1^i(t)/\lambda^*$  for the event at time  $t$ ; here it accepts three of the ones shown and rejects the others. In the third graph, the surviving type-1 events (purple squares) are interleaved with the surviving type-2 events (green pentagons). The next event is the earliest one among these surviving candidates. In practice, these sequences are constructed lazily so that we find only the earliest surviving event of each type. This is possible because the inter-arrival times between gold proposed events are distributed as  $\text{Exp}(\lambda^*)$ , making it straightforward to enumerate any finite prefix of a random infinite gold sequence.

---

**Algorithm 2** Data Simulation (thinning algorithm)

---

**Input:** interval  $[0, T]$ ; model parameters  
 $t_0 \leftarrow 0; i \leftarrow 1$   
**while**  $t_{i-1} < T$  :  $\triangleright$  draw event  $i$ , as it might fall in  $[0, T]$   
  **for**  $k = 1$  **to**  $K$  :  $\triangleright$  draw “next” event of each type  
    find upper bound  $\lambda^* \geq \lambda_k^i(t)$  for all  $t \in (t_{i-1}, \infty)$   
     $t \leftarrow t_{i-1}$   
    **repeat**  
      draw  $\Delta \sim \text{Exp}(\lambda^*)$ ,  $u \sim \text{Unif}(0, 1)$   
       $t \leftarrow t + \Delta$   $\triangleright$  time of next proposed event  
    **until**  $u\lambda^* \leq \lambda_k^i(t)$   $\triangleright$  accept proposal with prob  $\frac{\lambda_k^i(t)}{\lambda^*}$   
     $t_{i,k} \leftarrow t$   
     $t_i \leftarrow \min_k t_{i,k}; k_i \leftarrow \text{argmin}_k t_{i,k}$   $\triangleright$  earliest event wins  
     $i \leftarrow i + 1$   
**return**  $(k_1, t_1), \dots, (k_{i-1}, t_{i-1})$

---

Finally, how do we construct the upper bound  $\lambda^*$  on  $\lambda_k^i$ ? Recall that both of our self-modulating models (equations (3a) and (4a)) define  $\lambda_k^i = f_k(\tilde{\lambda}_k^i)$ , where  $f_k$  is monotonically non-decreasing. In both cases,  $\tilde{\lambda}_k^i$  is a sum of *bounded* functions on  $(t_{i-1}, \infty)$  (equations (3b) and (4)). In other words, we can express  $\tilde{\lambda}_k^i(t)$  as  $\mu + g_1(t) + \dots + g_n(t)$ . We can therefore replace each  $g$  function by its upper bound to obtain  $\lambda^* = f_k(\mu + \max_t g_1(t) + \dots + \max_t g_n(t))$ , in which the argument to  $f_k$  is a finite constant.

Specifically, in equation (3b), each summand  $\alpha_{k_h,k} \exp(-\delta_{k_h,k}(t - t_i))$  is upper-bounded by  $\max(\alpha_{k_h,k}, 0)$ . In equation (4), each summand  $w_{kd}h_d(t) = w_{kd} \cdot o_{id} \cdot (2\sigma(2c_d(t)) - 1)$  is upper-bounded by  $\max_{c \in \{c_{id}, \bar{c}_{id}\}} w_{kd} \cdot o_{id} \cdot (2\sigma(2c) - 1)$ . Note that the coefficients  $\alpha_{k_i,k}$  and  $w_{kd}$  may be either positive or negative.

While Algorithm 2 is classical and intuitive, we also implemented a more efficient variant. Instead of drawing the next event from each of  $K$  *different* non-homogeneous Poisson processes and keeping the earliest, we can construct a *single* non-homogenous Poisson process with aggregate intensity function  $\lambda^i(t) = \sum_{k=1}^K \lambda_k^i(t)$  over  $(t_{i-1}, \infty)$ . An upper bound  $\lambda^*$  on this aggregate function can be obtained by summing the upper bounds on the individual  $\lambda_k^i$  functions. We then use the thinning algorithm only to sample the next event time  $t_i$  from this aggregate process  $\lambda^i$ . Finally, we “dis-aggregate” by choosing  $k_i$  from the distribution  $p(k | t_i) = \lambda_k^i(t_i)/\lambda^i(t_i)$ .<sup>10</sup> This is equivalent to Algorithm 2. In terms of Figure 6, this more efficient version enumerates a gold sequence that is the union of the  $K$  gold sequences, and stops with the first accepted gold event. Thus, whereas Figure 6

<sup>10</sup>In practice, acceptance and disaggregation can be combined into a single step. That is, each successive event  $t$  proposed from the homogeneous Poisson( $\lambda^*$ ) process is either kept as type  $k$ , with probability  $\lambda_k^i(t)/\lambda^*$ , or rejected, with probability  $1 - \lambda^i(t)/\lambda^*$ . If it is accepted, we have found our next event  $(k_i, t_i)$ . If it is rejected, we increment  $t$  by  $\Delta \sim \text{Exp}(\lambda^*)$  to get the next proposed event.

DATASET	$K$	# OF EVENT TOKENS			SEQUENCE LENGTH		
		TRAIN	DEV	TEST	MIN	MEAN	MAX
SYNTHETIC	5	$\approx 480449$	$\approx 60217$	$\approx 60139$	20	$\approx 60$	100
RETWEETS	3	1739547	215521	218465	50	109	264
MEMETRACK	5000	93267	14932	15440	1	3	31
MIMIC-II	75	$\approx 1946$	$\approx 228$	$\approx 245$	2	4	33
STACKOVERFLOW	22	$\approx 343998$	$\approx 39247$	$\approx 97168$	41	72	736
FINANCIAL	2	$\approx 298710$	$\approx 33190$	$\approx 82900$	829	2074	3319

Table 1: Statistics of each dataset. We write “ $\approx N$ ” to indicate that  $N$  is the average value over multiple splits of one dataset (MIMIC-II, Stack Overflow, Financial Transaction); the variance is small in each such case.

DATASET	$K$	$D$	# OF MODEL PARAMETERS		
			SE-MPP	D-SM-MPP	N-SM-MPP
SYNTHETIC	5	256	55	60	922117
RETWEETS	3	256	21	24	921091
MEMETRACK	5000	64	50005000	50010000	702856

Table 2: Size of each trained model on each dataset. The number of parameters of neural Hawkes process is followed by the number of hidden nodes  $D$  in its LSTM (chosen automatically on dev data).

had to propose two type-1 events in order to get the first accepted type-1 event (the leftmost purple event), the more efficient version would not have had to spend time proposing either of those, because an earlier proposed event (the leftmost green event) had already been accepted and determined to be of type 2.

## C Experimental Details

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

### C.1 Dataset Statistics

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

### C.2 Training Details

We used a single-layer LSTM (Graves, 2012) in section 3.2.2, selecting the number of hidden nodes from a small set  $\{64, 128, 256, 512, 1024\}$  based on the performance on the dev set of each dataset. We empirically found that the model performance is robust to these hyperparameters.

When estimating integrals with Monte Carlo sampling,  $N$  is the number of sampled negative observations in Algorithm 1, while  $I$  is the number of positive observations. In practice, setting  $N = I$  was large enough for stable behavior, and we used this setting during training. For evaluation on dev and test data, we took  $N = 10 I$  for extra accuracy, or  $N = I$  when  $I$  was very large.

For learning, we used the Adam algorithm with its default settings (Kingma and Ba, 2015). Adam is a stochastic gradient optimization algorithm that continually adjusts the learning rate in each dimension based on adaptive estimates of low-order moments. Our training objective was unregularized log-likelihood.<sup>11</sup> We initialized the Hawkes process parameters and  $s_k$  scale factors to 1, and all other non-LSTM parameters (section 3.2.2) to small random values from  $\mathcal{N}(0, 0.01)$ . We performed early stopping based on log-likelihood on the held-out dev set.

<sup>11</sup> $L_2$  regularization did not appear helpful in pilot experiments, at least for our dataset size and when sharing a single regularization coefficient among all parameters.

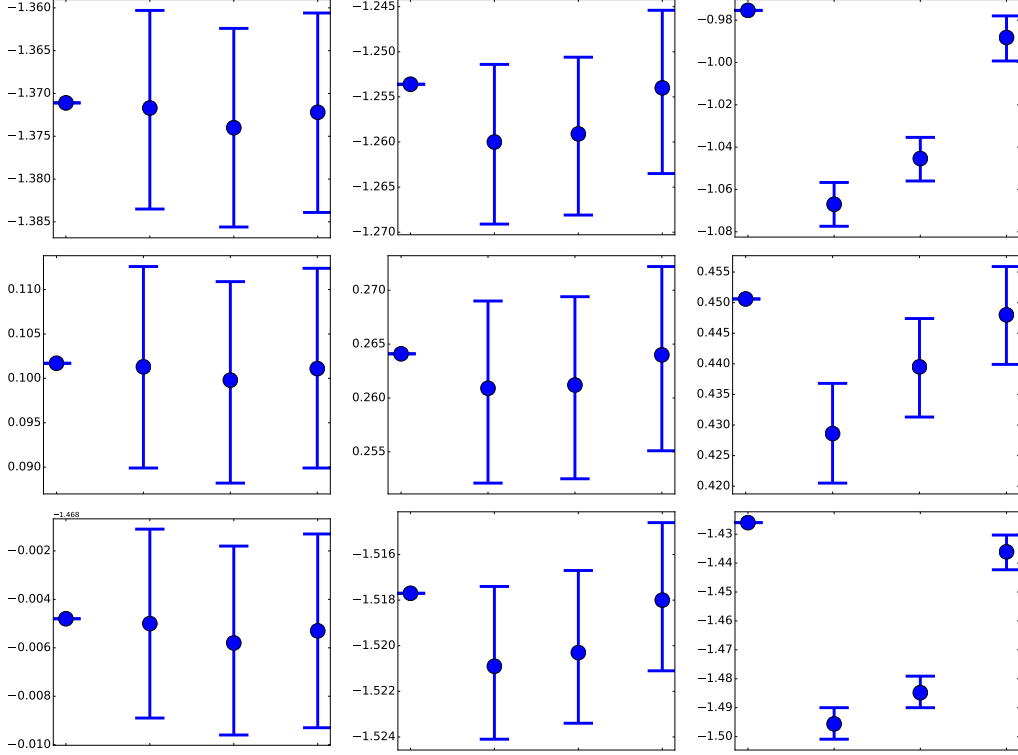


Figure 7: Log-likelihood (reported in nats per event) of each model on held-out synthetic data. Rows (top-down) are log-likelihood on the entire sequence, time interval, and event type. On each row, the figures (from left to right) are datasets generated by SE-MPP, D-SM-MPP and N-SM-MPP. In each figure, the models (from left to right) are Oracle, SE-MPP, D-SM-MPP and N-SM-MPP. Larger values are better. Note that log-likelihood for continuous variables can be positive, since it uses the log of a probability density that may be  $> 1$ .

### C.3 Model Sizes

The size of each trained model on each dataset is shown in Table 2. Our neural model has many parameters for expressivity, but it actually has considerably fewer parameters than the other models in the large- $K$  setting (MemeTrack).

### C.4 Pilot Experiments on Simulated Data

Our hope is that the neural Hawkes process is a flexible tool that can be used to fit naturally occurring data. As mentioned in section 6.1, we first checked that we could successfully fit data generated from *known* distributions. That is, when the generating distribution actually fell within our model family, could our training procedure recover the distribution in practice? When the data came from a decomposable process, could we nonetheless train our neural process to fit the distribution well?

We used the thinning algorithm (Appendix B.3) to sample event streams from different processes with randomly generated parameters: (a) a standard Hawkes process (SE-MPP, section 3.1), (b) our decomposable self-modulating process (D-SM-MPP, section 3.2.1), (c) our neural self-modulating processes (N-SM-MPP, section 3.2.2). We then tried to fit each dataset with all these models.<sup>12</sup>

The results are shown in Figure 7. We found that all models were able to fit the (a) and (b) datasets well with no statistically significant difference among them, but that the (c) models were substantially and significantly better at fitting the (c) datasets. In all cases, the (c) models were able to obtain a low KL divergence from the true generating model (the difference from the oracle column). This result suggests that the neural Hawkes process may be a wise choice: it introduces extra expressive power that is sometimes necessary and does not appear (at least in these experiments) to be harmful when it is not necessary.

<sup>12</sup>Details of data generation can be found in Appendix C.4.

We used Algorithm 2 to sample event streams from three different processes with randomly generated parameters: (a) a standard Hawkes process (SE-MPP), (b) our decomposable self-modulating process (D-SM-MPP), (c) our neural self-modulating processes (N-SM-MPP). We then tried to fit each dataset with all these models.

For each dataset, we took  $K = 5$  as the number of event types. To generate each event sequence, we first chose the sequence length  $I$  (number of event tokens) uniformly from  $\{20, 21, 22, \dots, 100\}$  and then used the thinning algorithm to sample the first  $I$  events over the interval  $[0, \infty)$ . For subsequent training or testing, we treated this sequence (appropriately) as the complete set of events observed on the interval  $[0, T]$  where  $T = t_I$ , the time of the last generated event. For each dataset, we generate 8000, 1000 and 1000 sequences for the training, dev, and test sets respectively.

For SE-MPP, we sampled the parameters as  $\mu_k \sim \text{Unif}[0.0, 1.0]$ ,  $\alpha_{j,k} \sim \text{Unif}[0.0, 1.0]$ , and  $\delta_{j,k} \sim \text{Unif}[10.0, 20.0]$ . The large decay rates  $\delta_{j,k}$  were needed to prevent the intensities from blowing up as the sequence accumulated more events. For D-SM-MPP, we sampled the parameters as  $\mu_k \sim \text{Unif}[-1.0, 1.0]$ ,  $\alpha_{j,k} \sim \text{Unif}[-1.0, 1.0]$ , and  $\delta_{j,k} \sim \text{Unif}[10.0, 20.0]$ . For N-SM-MPP, we sampled parameters from  $\text{Unif}[-1.0, 1.0]$ .

The results are shown in Figure 7, including log-likelihood (reported in nats per event) on the sequences and the breakdown of time interval and event types.

Another interesting question is whether the trained neural Hawkes model accurately predicts the real-valued *intensities*, since for the synthetic data we actually know the intensities. This is a more direct evaluation of whether the model is accurately recovering the dynamics of the underlying generative process. Here we compared only SE-MPP and N-SM-MPP.

All types behaved similarly, so we report only averages over the  $K$  types. For both processes (a) and (c), the true intensity’s variance was about 30% of the squared mean intensity. Thus, the intensity changes enough over time that predicting it at particular times is not a trivial challenge. To determine how well a model predicted the true intensity function, we measured the mean squared error (MSE) of predicted intensity at a large sample of times in the held-out test seqs, and report the MSE here as a percentage of the *variance* of the true intensity. By this construction, a simple baseline of predicting each event type’s mean intensity at all times would get 100% MSE.

Both the Hawkes and neural-Hawkes models predict the Hawkes intensities (a) accurately, at 1% MSE. This is similar to the leftmost column of Figure 7, where both models essentially achieved oracle performance. By contrast, for the complex neural Hawkes intensities (c), the neural Hawkes model achieves 9% MSE (still quite good) whereas Hawkes does far worse at 70% MSE. This is similar to the rightmost column of Figure 7, where the neural Hawkes model approached oracle performance but the Hawkes model did much worse.

### C.5 Retweet Dataset Details

The Retweets dataset (section 6.2) includes 166076 retweet sequences, each corresponding to some original tweet. Each retweet event is labeled with the retweet time relative to the original tweet creation, so that the time of the original tweet is 0. (The original tweet serves as the beginning-of-stream (BOS) marker as explained in Appendix A.2.) Each retweet event is also marked with the number of followers of the retweeter. As usual, we assume that these 166076 streams are drawn independently from the same process, so that retweets in different streams do not affect one another.

Unfortunately, the dataset does not specify the identity of each retweeter, only his or her popularity. To distinguish different kinds of events that might have different rates and different influences on the future, we divide the events into  $K = 3$  types: retweets by “small,” “medium” and “large” users. Small users have fewer than 120 followers (50% of events), medium users have fewer than 1363 (45% of events), and the rest are large users (5% events). Given the past retweet history, our model must learn to predict how soon it will be retweeted again and how popular the retweeter is (i.e., which of the three categories).

We randomly sampled disjoint train, dev and test sets with 16000, 2000 and 2000 sequences respectively. We truncated sequences to a maximum length of 264, which affected 20% of them. For computing training and test likelihoods, we treated each sequence as the complete set of events observed on the interval  $[0, T]$ , where 0 denotes the time of the original tweet (which is not included in the sequence) and  $T$  denotes the time of the last tweet in the (truncated) sequence.

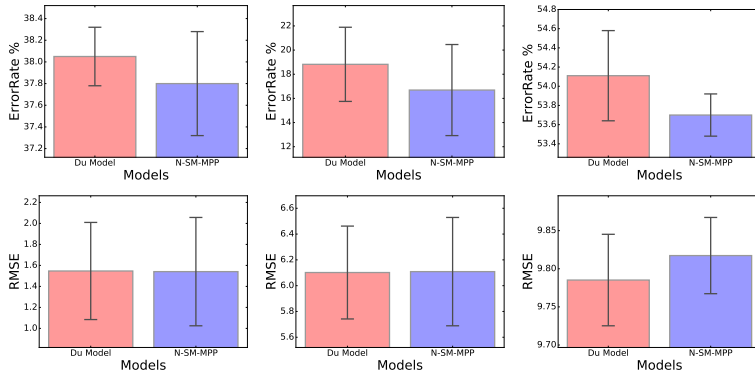


Figure 8: Prediction results on Financial Transactions, MIMIC-II, and Stack Overflow datasets (from left to right). Error bars show standard deviation over 5 experiments with different train-dev-test splits. For prediction of the types  $k_i$  (top row), our method achieved lower error in 4/5, 5/5, and 5/5 of the experiments. For prediction of the times  $t_i$  (bottom row), our method achieved lower error in 5/5, 2/5, and 0/5 of the experiments.

Figure 9 shows the learning curves of all the models, broken down by the log-probabilities of the event types and the time intervals separately. The scatterplot Figure 10 is a copy of Figure 3, and Figure 11 breaks down the log-likelihood by event type and time interval.

## C.6 MemeTrack Dataset Details

The MemeTrack dataset (section 6.2) contains time-stamped instances of meme use in articles and posts from 1.5 million different blogs and news sites, spanning 10 months from August 2008 till May 2009, with several hundred million documents.

As in Retweets, we decline to model the appearance of novel memes. Each novel meme serves as the BOS event for a stream of mentions on other websites, which we do model. The  $K$  event types correspond to the different websites. Given one meme’s past trajectory across websites, our model must learn to predict how soon it will be mentioned again and where.

We used the version of the dataset processed by Gomez Rodriguez et al. (2013), which selected the top 5000 websites in terms of the number of memes they mentioned. We truncated sequences to a maximum length of 32, which affected only 1% of them. We randomly sampled disjoint train, dev and test sets with 32000, 5000 and 5000 sequences respectively, treating them as before.

Because our current implementation does not allow for a marked BOS event (see Appendix A.2), we currently ignore where the novel meme was originally posted, making the unfortunate assumption that the stream of websites is independent of the originating website. Even worse, we must assume that the stream of websites is independent of the actual text of the meme. However, as we see, our novel models have some ability to recover from these forms of missing data.

Figure 12 shows the learning curves of the breakdown of log-likelihood with the same format as Figure 9. Figures 13 and 14 show the scatterplots in the same format as Figures 10 and 11.

## C.7 Prediction Task Details

Finally, we give further details of the prediction experiments from section 6.4. To avoid tuning on the test data, we split the original training set into a new training set and a held-out dev set. We train our neural model and that of Du et al. (2016) on the new training set, and choose hyper-parameters on the held-out dev set. Following Du et al. (2016), we consider three datasets, and use five different train-dev-test splits of each dataset to generate the experimental results in Figure 8. (None of the test sets’ examples were used during manual development of our system.)

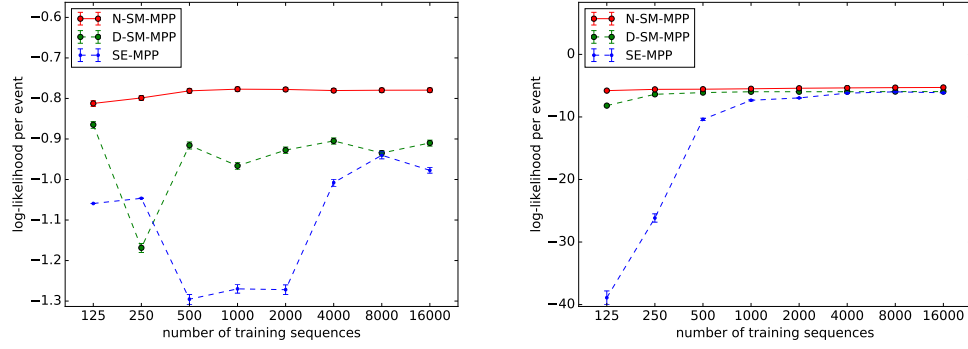


Figure 9: Learning curves (with 95% error bars) of all these models on the Retweets dataset, broken down by the log-probabilities of just the event types (left graph) and just the time intervals (right graph).

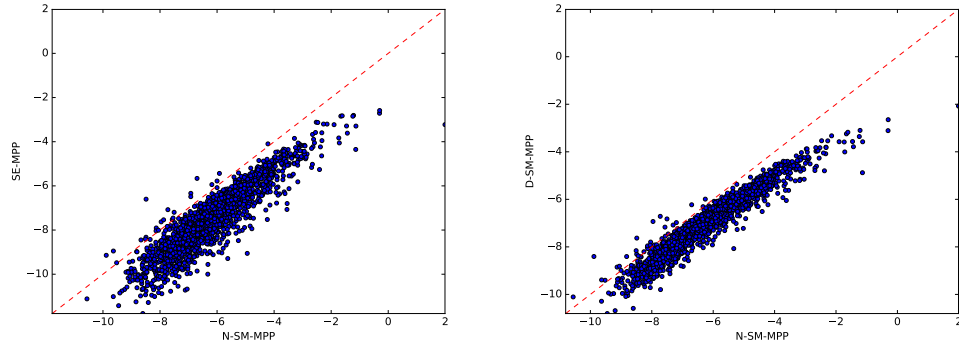


Figure 10: A larger copy of Figure 3, repeated here for convenience.

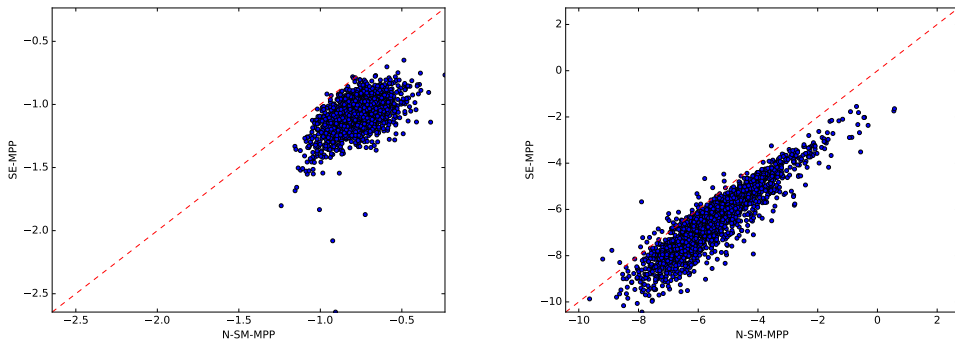


Figure 11: Scatterplots of N-SM-MPP vs. SE-MPP on Retweets. Same comparison as the left graph in Figure 10, but broken down by the log-probabilities of the event types (left graph) and the time intervals (right graph).

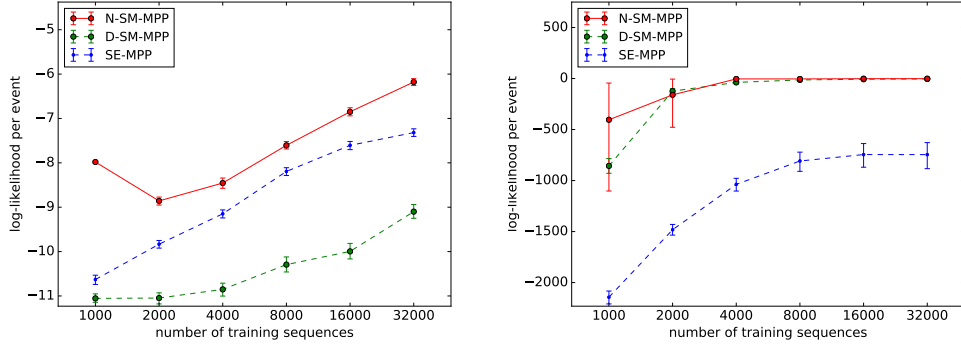


Figure 12: Learning curve (with 95% error bars) of all three models on the MemeTrack dataset, broken down by the log-probabilities of the event types (left graph) and the time intervals (right graph).

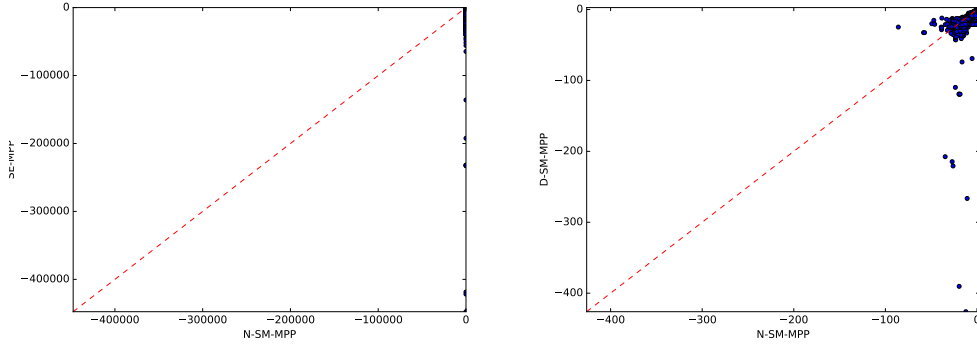


Figure 13: Scatterplot of N-SM-MPP vs. SE-MPP (left graph) and vs. D-SM-MPP (right graph) on MemeTrack. N-SM-MPP outperforms D-SM-MPP on 93.02% of the test sequences. This is not obvious from the plot, because almost all of the 5000 points are crowded near the upper right corner. Most of the visible points are outliers where N-SM-MPP performs unusually badly—and D-SM-MPP typically does even worse.

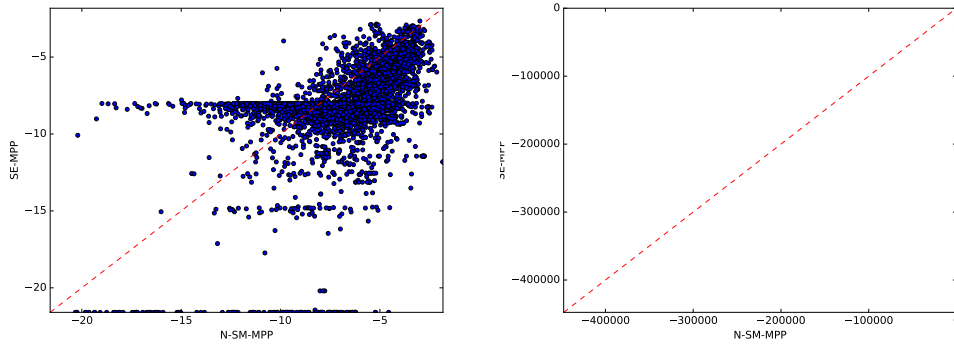


Figure 14: Scatterplots of N-SM-MPP vs. SE-MPP on MemeTrack. Same comparison as the left graph of Figure 13, but broken down by the log-probabilities of the event types (left graph) and the time intervals (right graph).

## D Ongoing and Future Work

We are currently exploring several extensions to deal with more complex datasets. Based on our survey of existing datasets, we are particularly interested in handling:

- immediate events ( $t_{i-1} = t_i$ ), as discussed in footnote 1
- “baskets” of events (several events that are recorded as occurring simultaneously but without a specified order, e.g., the purchase of an entire shopping cart)
- hard constraints on the event type sequence  $k_1, k_2, \dots$
- marked events<sup>13</sup> and annotated events<sup>14</sup>
- causation by external events (artificial clock ticks, periodic holidays, weather)
- richer drift functions<sup>15</sup>
- hybrid of D-SM-MPP and N-SM-MPP, allowing direct influence from past events
- multiple agents each with their own state, who observe one another’s actions (events)

More important, we are interested in modeling causality. The current model might pick up that a hospital visit elevates the instantaneous probability of death, but this does not imply that a hospital visit *causes* death. (In fact, the severity of an earlier illness is usually the cause of both.)

A model that can predict the result of interventions is called a causal model. Our model family can naturally be used here: any choice of parameters defines a generative story that follows the arrow of time, which can be interpreted as a causal model in which patterns of earlier events *cause* later events to be more likely. Such a causal model predicts how the distribution over futures would change if we intervened in the stream of events.

In general, one cannot determine the parameters of a causal model based on purely observational data (Pearl, 2009). Thus, in future, we plan to determine such parameters through randomized experiments by deploying our model family as an environment model within reinforcement learning. A reinforcement learning agent *tests* the effect of random interventions to discover their effect (exploration) and thus orchestrate more rewarding futures (exploitation).

In our setting, the agent is able to stochastically insert or suppress certain event types and observe the effect on subsequent events. Then our LSTM-based model will discover the causal effects of such actions, and the reinforcement learner will discover what actions it can take to affect future reward. Ultimately this could be a vehicle for personalized medical decision-making. Beyond the medical domain, a quantified-self smartphone app may intervene by displaying fine-grained advice on eating, sleeping, exercise, and travel; a charitable agency may intervene by sending a social worker to provide timely counseling or material support; a social media website may increase positive engagement by intelligently distributing posts; or a marketer may stimulate consumption by sending more targeted advertisements.

<sup>13</sup>A “mark” is some structured data attached to an event: for example, the textual content associated with a tweet, or the medical records associated with a doctor visit. The model should predict the marks from each event and its underlying hidden state, and they should be fed back into the LSTM as additional input.

<sup>14</sup>Humans may be asked to classify the events in an event stream or the relationships among its events. Unlike marks, these annotations are not involved in the process that generates the event stream, and so are not fed into the LSTM as input. Rather, they are assumed to be generated *post hoc* by the human from the entire observed stream—and may depend on the human’s implicit reconstruction of the hidden states. We can use any available annotations to help reconstruct the hidden states (Zaidan and Eisner, 2008), if we model them as stochastic functions of the hidden states. In particular, annotations on the training data serve as side information to improve training of the model. As a simple example, an annotation of the training event  $(k_i, t_i)$  could be assumed to depend also on the subsequent LSTM state  $\mathbf{h}(t_i^+) \stackrel{\text{def}}{=} \lim_{t \rightarrow t_i^+} \mathbf{h}(t)$ .

<sup>15</sup>We expect the exponential drift in equation (7) to be expressive enough in most settings. In principle, however, one might want to allow periodic fluctuation of the intensity between events, say by using a *complex* exponential in (7). Another way to increase expressivity would be to compute drift using the LSTM itself, by injecting special “clock tick” events into the input stream at regular intervals (compare Xiao et al., 2017b). Each clock tick event  $(k_i, t_i)$  causes a rich nonlinear update of the LSTM state via equations (5)–(6), except that it should always set  $\mathbf{c}_{i+1} = \mathbf{c}(t_i)$  for continuity. In this design, the interval between ordinary events is modeled piecewise—it is divided up into short pieces by the clock ticks, with  $\mathbf{c}(t)$  on each piece modeled using our current function family.