Learning Neural Point Processes with Latent Graphs

Qiang Zhang
University College London
London, United Kingdom
qiang.zhang.16@ucl.ac.uk

Aldo Lipani
University College London
London, United Kingdom
aldo.lipani@ucl.ac.uk

Emine Yilmaz
University College London & Amazon
London, United Kingdom
emine.yilmaz@ucl.ac.uk

ABSTRACT

Neural point processes (NPPs) employ neural networks to capture complicated dynamics of asynchronous event sequences. Existing NPPs feed all history events into neural networks, assuming that all event types contribute to the prediction of the target type. However, this assumption can be problematic because in reality some event types do not contribute to the predictions of another type. To correct this defect, we learn to omit those types of events that do not contribute to the prediction of one target type during the formulation of NPPs. Towards this end, we simultaneously consider the tasks of (1) finding event types that contribute to predictions of the target types and (2) learning a NPP model from event sequences. For the former, we formulate a latent graph, with event types being vertices and non-zero contributing relationships being directed edges; then we propose a probabilistic graph generator, from which we sample a latent graph. For the latter, the sampled graph can be readily used as a plug-in to modify an existing NPP model. Because these two tasks are nested, we propose to optimize the model parameters through bilevel programming, and develop an efficient solution based on truncated gradient back-propagation. Experimental results on both synthetic and real-world datasets show the improved performance against state-of-the-art baselines. This work removes disturbance of non-contributing event types with the aid of a validation procedure, similar to the practice to mitigate overfitting used when training machine learning models.

CCS CONCEPTS

• **Information systems** → **Web mining**; **Data streaming**.

KEYWORDS

Point Process, Latent Graph, Bilevel Programming

1 INTRODUCTION

Many types of events occur asynchronously in the world. There is a proliferation of research interest in capturing their temporal dynamics so as to predict *which type* of events will happen next and *when*. A *de facto* mathematical tool to model such events is temporal point process [10, 13]. This finds applications in many domains such as financial analysis [3], social networks [20] and clinical visiting [49]. Here, these applications are characterized by the presence of (a) more than one *event type*, and (b) events that can mutually influence themselves, i.e., the occurrence of one type of event at a certain time can cause or prevent the occurrence of future events of the same or another type. Discovering correlation and dependency of events is important in the prediction and explainability of future asynchronous events.

Various neural network-based studies have explored the complex impact of historical events on the occurrence of future ones [16, 39, 42, 46, 54]. These works feed historical events into a neural network to compute either a conditional intensity, a cumulative intensity or a conditional probability. One major issue is that these neural point processes (NPPs), when predicting the occurrence of an event type, use all types of historical events before any given moment, failing to consider some event types do not actually contribute to predicting that event type. This failure hurts NPP as non-contributing types can be disturbance. Therefore, it is important to remove such disturbance from the formulation of NPPs.

To address the aforementioned issue, we learn to omit non-contributing event types to remove their disturbance on TPPs when predicting the target types. Specifically, we consider the following two tasks: (1) finding the set of event types that contribute to the prediction of one target type, and (2) learning NPP from event sequences. Since the contributing relationship is pairwise and not mutually exchangeable, we formulate a latent graph, with event types as vertices and binary contributing values as directed edges. We develop a probabilistic graph generator based on the Random Graph theory. To learn NPP, we use the generated graph to modify a NPP model by masking off certain events whose types do no contribute to the target type. Because these two tasks are nested, we formulate them as a bilevel programming problem and develop an efficient solution based on truncated gradient back-propagation. The bilevel programming with the aid of the training-validation procedure aims to minimize the overall generalization error. The contributions of this paper are:

- We learn to omit non-contributing event types for the prediction of the target types during learning NPPs. The contributing relationships among event types constitute a latent graph; to sample a graph, we develop a probabilistic graph generator based on Random Graph. The generated graph modifies the formulation of NPP.
- We formulate the two tasks, i.e., learning NPP and finding the set of contributing event types, as a bilevel programming problem. An efficient solution is also provided to optimize the model parameters.
- We conduct systematic experiments on both synthetic and real-world datasets to show the superiority of our model against state-of-the-art baselines.

The reminder of the paper goes as follows: § 2 introduces the used notations; § 3 presents preliminary concepts about temporal point process, and introduces Random Graph and bilevel programming that help formulate our idea; § 4 details how to formulate a NPP and

WWW '21, April 19–23, 2021, Ljubljana, Slovenia
2021. ACM ISBN 978-1-4503-8312-7/21/04.
https://doi.org/10.1145/3442381.3450135

employ the bilevel programming solution; § 5 describes the used datasets and experimental setup; § 6 is devoted to experimental results; § 7 summarizes related works and § 8 concludes the paper.

2 NOTATION

In this section we introduce the notation used throughout the paper.

Symbol	Description
$\mathcal{U}$	a set of event types.
$\mathcal{S}$	an event sequence.
t	the time of an event.
u, v	the type of an event.
i, j	the order number of an event in a sequence.
$N_u(t)$	the counting process for the events of type u.
$\mathcal{H}_t$	the set of events that happened before time t.
$\lambda^*(t)$	the conditional intensity function.
$p^*(t)$	the conditional probability density function.
$F^*(t)$	the cumulative distribution function.

3 PRELIMINARIES

3.1 Temporal Point Processes

A temporal point process [13, 14] is a stochastic process that models a sequence of events in a continuous time domain. It can be equivalently represented as a counting process $N(t)$, which records the number of events that have happened before time t. A multivariate point process describes the temporal evolution of multiple event types $\mathcal{U}$. Let $\mathcal{S} = \{(v_i, t_i)\}_{i=1}^{L}$ be an event sequence, where the tuple (v_i, t_i) is the i-th event, $v_i \in \mathcal{U}$ and $t_i \in \mathbb{R}$ are its event type and timestamp respectively. Let $\mathcal{H}_t := \{(v', t') \,|t' < t, v' \in \mathcal{U}\}$ be a sequence of historical events that occurred prior to t.

Hawkes Processes. The history-dependent assumption of point processes translates into an intensity function $\lambda^*(t)$ defined as:

$$\lambda^*(t) = \mu + \sum_{(v',t') \in \mathcal{H}_t} \phi(t - t'), \tag{1}$$

where $\mu \geq 0$ indicates the base intensity independent of the history, while $\phi(t) > 0$ is a triggering kernel measuring the event peer influence. To emphasize the influence between various types of events, we rewrite $\phi(t)$ as $\phi_{u,v}(t)$, which means a type-v history event increases the intensity of a subsequent type-u event [19]. In this example, the occurrence of a past type-v event increases the intensity function $\phi_{u,v}(t - t')$ for $0 < t' < t$. One of our tasks is to learn $\lambda^*(t)$ of NPP from event sequences. Based on the intensity function, it is straightforward to derive the probability density function $p^*(t)$ and the cumulative distribution function $F^*(t)$ [43].

$$p^*(t) = \lambda^*(t) \exp\left(-\int_{t_{i-1}}^{t} \lambda^*(\tau) \mathrm{d}\tau\right), \tag{2}$$

$$F^*(t) = 1 - \exp\left(-\int_{t_{i-1}}^{t} \lambda^*(\tau) \mathrm{d}\tau\right). \tag{3}$$

Optimization. The parameters of point processes can be learnt by Maximum Likelihood Estimation (MLE). Other advanced and more complex adversarial learning [51] and reinforcement learning [31] methods have been proposed, however we use MLE for its simplicity. To apply MLE, a loss function is derived based on the negative log-likelihood. The log-likelihood of $\mathcal{S}$ is defined as:

$$\begin{aligned}
\mathcal{L} &= \log\left[\prod_{i=1}^{L} p(t_i) p(u = v_i | t_i)\right] \\
&= \log\left[\prod_{i=1}^{L} \lambda(t_i) \exp\left(-\int_{t_{i-1}}^{t_i} \lambda(\tau) \mathrm{d}\tau\right) \frac{\lambda_{v_i}(t_i)}{\lambda(t_i)}\right] \\
&= \sum_{i=1}^{L} \log \lambda_{v_i}(t_i) - \int_0^T \lambda(\tau) d\tau
\end{aligned} \tag{4}$$

3.2 Random Graph

Random Graph (RG) is an elegant mathematical tool to work with probability distributions over graphs [8]. It plays a role at the intersection of probability theory and graph theory. By applying probability distribution and random processes, RG aims to study when certain properties of typical graphs are likely to arise [7]. The general idea is to, given a set of isolated vertices, successively and randomly add edges between these vertices. RG has been found helpful to construct complex networks [33, 34].

The Erdös–Rényi (ER) Model. A variety of RG models have been proposed to generate distributions over graphs. One of the most commonly studied is the Erdös–Rényi (ER) model [18], which assumes all graphs have the same probability to occur given a fixed set of vertices and number of edges. Most closely, Gilbert [25] propose $G(n, p)$, assuming that each edge between any pairs of the n vertices exists independently with a fixed probability $0 < p < 1$. With $G(n, p)$, a graph is constructed by randomly connecting pairs of vertices. The probability of one particular graph with m edges is $p^m(1 - p)^{N-m}$, where $N = \binom{n}{2}$. Equivalently, each edge is an independent Bernoulli random variable with the same probability.

3.3 Bilevel Programming

Bilevel programming specifies an optimization problem of two tasks with a hierarchical relationship. More specifically, variables of one task are nested to an optimal solution to the other task. The general formulation of a bilevel programming problem consists of two parts:

$$\min_{\theta \in \Theta, \psi} F(\theta, \psi) \tag{5a}$$

$$\text{s.t. } \psi \in \arg\min_{\psi' \in \Psi} f(\theta, \psi') \tag{5b}$$

where the functions $F : \mathbb{R}^{n_1} \times \mathbb{R}^{n_2} \to \mathbb{R}$ and $f : \mathbb{R}^{n_1} \times \mathbb{R}^{n_2} \to \mathbb{R}$ are the upper-level and lower-level task objectives. Similarly, $\theta \in \Theta \subset \mathbb{R}^{n_1}$ is the upper-level variable and $\psi \in \Psi \subset \mathbb{R}^{n_2}$ is the lower-level variable. More details can be found in reference [12].

Bilevel programming has been studied in numerous areas. Historically, it is closely related to the game theory [47] in economics Recently, bilevel programming has contributed to various ML tasks including model selection [5], multi-task learning [21] and adversarial training [40], and hyper-parameter tuning [22].

4 METHOD

To emphasize the effects between types u and v, we factorize the kernel $\phi_{u,v}(t)$ as

$$\phi_{u,v}(t) = \mathcal{G}_{u,v} \cdot \kappa(t). \tag{6}$$

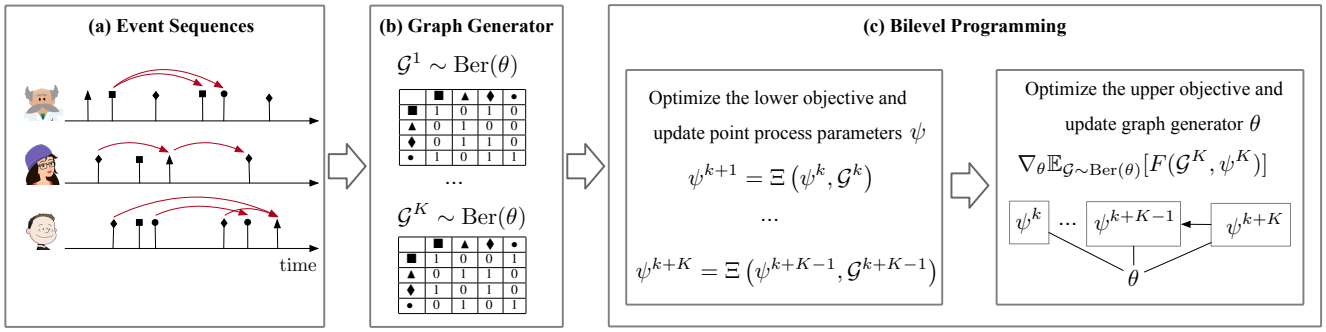


Figure 1: The schematic representation of learning neural point process from observed event sequences.

The first factor $\mathcal{G}_{u,v} \in \{0,1\}$ indicates whether type-v events contribute to predicting type-u events; a matrix $\mathcal{G} = [\mathcal{G}_{u,v}] \in \{0,1\}^{|\mathcal{U}|\times|\mathcal{U}|}$. The second factor is a *kick* function $\kappa(t)$ that characterizes the time-decaying influence of one historical type-v event on the occurrence of one future type-u event.

We find a $\mathcal{G}$ revealing non-contributing types while simultaneously learning $\kappa(t)$ from event sequences. Towards this end, we propose a probabilistic way to generate the matrix $\mathcal{G}$. As the probabilistic generator is nested with the kick function, we formulate this procedure as bilevel programming: the upper-level task is to generate the graph $\mathcal{G}$ while the lower-level task is to learn the kick function for the prediction of sequential data. Fig. 1 is a schematic illustration of the proposed model. The graph generator takes event sequences as input to calculate the graph distribution, from which we sample a latent graph and use it to modify a NPP model. The loss minimization of NPP via bilevel programming forces to discover the optimal graph $\mathcal{G}$ for event prediction.

4.1 Probabilistic Graph Generator

This section presents a Probabilistic Graph Generator (PGG) that uncovers the latent graph from observed event sequences. We use $\mathcal{G} = (\mathcal{U}, \mathcal{E})$ to denote a graph (matrix), where $\mathcal{U}$ is the set of nodes and $\mathcal{E} = \{\mathcal{G}_{u,v} | \mathcal{G}_{u,v} = 1\}$ is the set of edges. For sake of consistency, we use $\mathcal{G}$ to represent the graph. Generating the graph $\mathcal{G}$ involves discrete variables, which hinders continuous gradient-based optimization. An alternative is to parameterize a probability distribution over graphs. Therefore we propose a probabilistic graph generator (PGG). We first develop a analytical basis of the distributions of graph structures based on Random Graph, then describes how to sample from the graph distribution.

Since the graph is latent, we reconstruct edges based on Random Graph. One of the most commonly studied RG models is the Erdös–Rényi model, which assumes that each edge is generated independently with a same probability [25]. While for the graph, we assume the existence of an edge is merely decided by the pair of involved event types, with different probabilities. Based on this assumption, we consider $\mathcal{G}_{u,v}$ to be conditionally independent given the vertices u and v. In the regime of Bayesian models, this is equivalent to assuming the same prior on all possible structures.

Because a contributing relationship is assumed to be determined only by event types, we employ type embeddings to compute the probability of the existence of directed edges. We use an embedding matrix E to obtain dense type embeddings from the one-hot form, $\boldsymbol{tp}_u = E * \text{one\_hot}(u)$ where one_hot (u) is the one-hot form representation of the type u, $\boldsymbol{tp} \in \mathbb{R}^{|\mathcal{U}|*d}$ is the dense type embedding and d is the dimension of type embeddings. For the event type u and v, we compute type embeddings $\boldsymbol{tp}_u$ and $\boldsymbol{tp}_v$ with an embedding matrix E. Hence the probability that type-v contributes to predicting the type-u is computed as:

$$\theta_{u,v} = \rho e^{-\left\|\boldsymbol{tp}_u * W_{\mathcal{G}} * \boldsymbol{tp}_v^{\mathrm{T}}\right\|}. \tag{7}$$

Here, $W_{\mathcal{G}} \in \mathbb{R}^{d*d}$ denotes a matrix of trainable parameters, which breaks the equality between $\theta_{u,v}$ and $\theta_{v,u}$; ρ is a hyper-parameter that controls the graph sparsity. A more complicated model such as neural networks could be tried but we empirically find Equation 7 works well possibly because of the smaller number of parameters. Then, we use a Bernoulli distribution to indicate type-v contributes to predicting type-u

$$\mathcal{G}_{u,v} \sim \mathrm{Ber}(\theta_{u,v}). \tag{8}$$

By modeling all possible edges as a set of mutually independent Bernoulli random variables with parameter matrix θ we can sample a graph as $\mathcal{G} \sim \mathrm{Ber}(\theta)$. One problem with the sampling of Bernoulli distribution is the inability to gradient-based end-to-end training. To solve this problem, we apply the Gumbel-Max trick that provides a simple and efficient way to draw samples from a Bernoulli distribution [37].

Suppose the graph space is denoted as $\mathbb{G} = \{\mathcal{G}\}$. The the number of edges increase quadratically with the number of nodes $|\mathcal{U}|$, and the graph space $\mathbb{G}$ expands as $2^{|\mathcal{U}|\times|\mathcal{U}|}$. We analyze our model owns an advantage of good linear scalability compared to counterparts that search the whole exponentially large set of graph structures with differentiable optimization, such as reinforcement learning [56]. This can be explained by facts: (1) the graph generator calculates the existence of an edge merely based on the two involved nodes; and (2) the parameterization of Bernoulli variables enables one to sample a new graph in the space complexity of $O(|\mathcal{U}|^2)$ and trivial time complexity due to the merit of parallel sampling.

4.2 Modify NPPs with Dependency Graph

We now describe how to use the sampled graph $\mathcal{G}$ to modify a NPP. As an event consists of its type and timestamp, we represent the event (v_i, t_i) $\boldsymbol{x}_i = \boldsymbol{tp}_v + \boldsymbol{pe}_{(v_i,t_i)}$, where $\boldsymbol{pe}$ is the positional

embedding. We choose to modify the SAHP model by [54] because: (1) the factor $\mathcal{G}_{u,v}$ can be easily used as a plug-in to remove certain types of events from history if these types do not contribute to the predictions of another event type, and (2) it is the state-of-the-art point process model. Specifically, we modify the conventional way of computing attention scores, i.e., embedded Gaussian, by multiplying $\mathcal{G}_{u,v} \in \{0, 1\}$:

$$f(\boldsymbol{x}_{i+1}, \boldsymbol{x}_j) = \mathcal{G}_{u,v} \cdot \exp\left(\boldsymbol{x}_{i+1}\boldsymbol{x}_j^T\right), \tag{9}$$

where u and v are the types of the $(i+1)$-th and j-th event. $\mathcal{G}_{u,v}$ hinders historical influence of type-v events if type-v does not contribute to predicting type-u. The rest of how to compute the intensity is the same as [54]. The attention scores $f(\cdot, \cdot)$ is used to calculate a hidden embedding $\boldsymbol{h}_{u,i+1}$ that summarizes influence of history events before t_{i+1} on the type-u event at the $(i+1)$-th position of a sequence, where the function $g(\cdot)$ is a linear transformation. $\boldsymbol{h}_{u,i+1}$ is used to calculate three dynamic parameters $\mu_{u,i+1}$, $\eta_{u,i+1}$ and $\gamma_{u,i+1}$. The three parameters are used to compute the intensity function, and the objective function of the event sequence. Details are in the Appendix.

Remark. SAHP uses the attention mechanism to measure contributions of each event type. This model holds the underlying assumption that it is beneficial to consider all types when predicting the target one. However, as we explained in the Introduction section, this assumption is problematic because some event types may not contribute. Adding $G_{u,v}$ in Eq.(9) provides the option to omit those non-contributing event types so as to avoid their disturbance. In this way, the learned attention is a more accurate measurement of contributions.

We underline the graph does not represent a causal structure as the objective, unlike the penalized log-likelihood such as Bayesian Information Criterion, focuses on predictive performance. The latent graph is most similar to Granger causality (G-causality). From the viewpoint of temporal point processes, G-causality aims to identify a subset of event types $V \subseteq U$ for the type-u event, such that $\lambda_u(t)$ only depends on historical events of types in V. The fundamental difference is that G-causality is deterministic while the latent graph in this paper is stochastic.

4.3 Bilevel Programming

Type embeddings are used to (1) generate the graph $\mathcal{G}$ in Equation 7 and (2) calculate the influence weights of historical events with Equation 15. Direct optimization of Equation 4 can be infeasible: gradient descents of NPPs can change type embedding matrix E, which can change the graph $\mathcal{G}$ generated by PGG and further the landscape of the NPP loss function; since the gradients are calculated based on the losses, the changes in the landscape mean that the previous gradients are no longer aligned to the descent direction of the actual loss. Because the embedding matrix E of NPPs are nested to an optimal solution of the PGG, bilevel programming is a natural fit to find the optimum.

Objective Formulation. Bilevel programming involves two nested tasks. Our ultimate goal is to find the graph $\mathcal{G} \in \mathbb{G}$ that minimizes the generalization error of the point process, therefore the minimization of this error should play a leading role. We formulate the bilevel programming as

$$\min_{\theta\in\Theta,\psi} \mathbb{E}_{\mathcal{G}\sim\mathrm{Ber}(\theta)}[F(\mathcal{G}, \psi)], \tag{10a}$$

$$\text{s.t. } \psi \in \arg\min_{\psi'\in\Psi} \mathbb{E}_{\mathcal{G}\sim\mathrm{Ber}(\theta)}[f(\mathcal{G}, \psi')]. \tag{10b}$$

Eq. (10a) and Eq. (10b) are the upper and lower objective function respectively; the upper-level variables are the parameter θ (actually the parameter $W_{\mathcal{G}}$ as shown in Equation 7) of the graph generator while the lower-level variables are the parameter ψ of the NPP, including the embedding matrix E and other trainable parameters in the conventional NPP models.

The upper objective aims to find an optimal binary structure $\mathcal{G}$ while the lower objective aims to learn a point process model given $\mathcal{G}$. As the upper objective is to minimize the generalization error, it can be implemented as the loss on the validation subset D_{val}. We therefore set out to use the training dataset D_{tr} to learn the kick function for a fixed dependency graph, and use the validation D_{val} to find the optimal graph.

$$F(\mathcal{G}, \psi) = -\sum_{\mathcal{S}\in D_{\mathrm{val}}} \mathcal{L}(\mathcal{S}|\mathcal{G}, \psi) \tag{11a}$$

$$f(\mathcal{G}, \psi) = -\sum_{\mathcal{S}\in D_{\mathrm{tr}}} \mathcal{L}(\mathcal{S}|\mathcal{G}, \psi) \tag{11b}$$

where $\mathcal{L}$ is the log-likelihood function of event sequences defined in Eq. (4). The minimization of the negative log-likelihood forces to discover the optimal graph $\mathcal{G}$ for event prediction. Equivalently, this model uses a validation set for the sake of generalization.

The resulting bilevel problem is difficult to solve efficiently for the following two reasons: 1) the solution of Eq. (10b) cannot be written in a closed form due to the non-convex objective; and 2) the intractability of the expectation: $\mathbb{E}_{\mathcal{G}\sim\mathrm{Ber}(\theta)}[f(\mathcal{G}, \psi)] = \sum_{\mathcal{G}\in\mathbb{G}} p_\theta(\mathcal{G})\mathcal{L}(\mathcal{S}|\mathcal{G}, \psi)$ since it is composed of $2^{|\mathcal{U}|\times|\mathcal{U}|}$ cases. Therefore, we now develop an efficient algorithm to find approximated solutions.

Efficient Optimization. Bilevel programming can be tackled via replacing the minimization of f with repeated applications of an iterative dynamics Ξ [15]. One common approach implements Ξ as iterative gradient descents [22, 36]. Let $\psi^{(K)}$ indicate the lower variables after K iterations of the dynamics Ξ, that is, $\psi^{(K)} = \Xi(\psi^{(K-1)}, \theta) = \Xi(\Xi(\psi^{(K-2)}, \theta), \theta)$ and so on.

Assuming that, given any θ, the optimal solution to the lower-level objective is unique, we define ψ as an implicit function $\psi(\theta)$ of θ, then bilevel programming reduces to find the optimal upper variable θ [12]. If θ and ψ are real numbers and the functions and dynamics are differentiable, we can compute the gradient of the upper function $F(\theta, \psi^K)$ w.r.t. θ using the chain rule [23]:

$$\nabla_\theta F(\theta, \psi^K) = \partial_\theta F\left(\theta, \psi^K\right) + \partial_\psi F\left(\theta, \psi^K\right) \nabla_\theta \psi^K, \tag{12}$$

where ∇ indicates the gradient and ∂ the partial derivative. $\nabla_\theta F(\theta, \psi^K)$ is called hyper-gradient. The right-hand side of this equation requires unrolling the dynamics repeatedly $\psi^{k+1} = \Xi(\psi^k, \theta)$ with $1 \le k \le K$, and can be implemented efficiently using reverse-mode algorithmic differentiation [26].

To solve the bilevel optimization problem efficiently, we develop a solution based on iterative gradient descents [44]. Specifically,

Algorithm 1: The algorithm to learn a NPP.

input : $D_{\text{tr}}, D_{\text{val}}, \rho, \eta_1, \eta_2, K$

begin
  Initialize parameters
  while *Stopping condition is not met* **do**
    $k \leftarrow 1, \theta \leftarrow \text{PGG}(W, \rho)$
    while *Lower objective decreases* **do**
      $\mathcal{G}^k \sim \text{Ber}(\theta)$ // Sample a graph
      $f(\mathcal{G}^k, \psi^k) = \sum_{\mathcal{S} \in D_{\text{tr}}} \mathcal{L}(\mathcal{S}|\mathcal{G}^k, \psi^k)$
      $\psi^{k+1} = \Xi\left(\psi^k, \mathcal{G}^k\right)$ // Optimize the lower objective function
      $k \leftarrow k + 1$
      if $k = K$ **then**
        $\mathcal{G}^k \sim \text{Ber}(\theta)$
        $F(\mathcal{G}^k, \psi^k) = \sum_{\mathcal{S} \in D_{\text{val}}} \mathcal{L}(\mathcal{S}|\mathcal{G}^k, \psi^k)$.
        $p^k \leftarrow \partial_\psi F\left(\mathcal{G}^k, \psi^k\right), q^k \leftarrow \partial_{\mathcal{G}} F\left(\mathcal{G}^k, \psi^k\right)$ // Compute the hyper-gradients
        for *j from* $k-1$ *to* $k-K$ **do**
          $\mathcal{G}^j \sim \text{Ber}(\theta)$
          $p^j \leftarrow p^{j+1} \partial_\psi \Xi\left(\psi^j, \mathcal{G}^j\right)$, $q^j \leftarrow q^{j+1} + p^j \partial_{\mathcal{G}} \Xi\left(\psi^j, \mathcal{G}^j\right)$
        $\theta \leftarrow \theta - \eta_2 q^j$ // Optimize the upper objective function
        $W_{\mathcal{G}} \leftarrow W_{\mathcal{G}} - \eta_2 \nabla_{W_{\mathcal{G}}} \theta$
        $k \leftarrow 1$
  return θ, ψ

we implement the learning dynamics based on the principle of stochastic gradient descent (SGD):

$$\psi^{k+1} = \Xi\left(\psi^k, \mathcal{G}^k\right) = \psi^k - \eta \nabla_\psi f\left(\mathcal{G}^k, \psi^k\right), \tag{13}$$

where η is the learning rate and $\mathcal{G}^k \sim \text{Ber}(\theta)$ is the k-th drawn with $1 \le k \le K$. This leads to ψ convergence because it depends on θ when $K \to \infty$ [9]. Given that ψ^K is the approximation of the optimal solution of the lower-objective Eq. (10b), we need to estimate the hyper-gradient for the upper-objective Eq. (10a) to update θ:

$$\begin{aligned}&\nabla_\theta \mathbb{E}_{\mathcal{G} \sim \text{Ber}(\theta)}\left[F\left(\mathcal{G}^K, \psi^K\right)\right] = \mathbb{E}_{\mathcal{G} \sim \text{Ber}(\theta)}\left[\nabla_\theta F\left(\mathcal{G}^K, \psi^K\right)\right] \\ =&\mathbb{E}_{\mathcal{G} \sim \text{Ber}(\theta)}\left[\partial_{\mathcal{G}} F\left(\mathcal{G}^K, \psi^K\right) \nabla_\theta \mathcal{G} + \partial_\psi F\left(\mathcal{G}^K, \psi^K\right) \nabla_\theta \psi^K\right].\end{aligned} \tag{14}$$

The swap of expectation and gradient is based on the finite number of random variables and a bounded F. Note that $\nabla_\theta \mathcal{G}$ occurs in both Equation 13 (through $\nabla_\theta \Xi\left(\psi^k, \mathcal{G}^k\right)$) and Equation 14. We use the so-called straight-through estimator [4], setting $\nabla_\theta \mathcal{G} := I$. Finally, we take the single sample Monte Carlo estimator of Eq. (14) to update the parameters θ as also done by [23]. Algorithm 1 describes how to solve the bilevel optimization problem. Shaban et al. [44] analyze that the bias of the K-iteration gradient as the approximation decays exponentially in K. This exponential decaying property guarantees the convergence of bilevel programming to update the upper variables with truncated back-propagation. More convergence analysis can be found in [44].

Computational Complexity. Conventional NPPs, like SAHP [54], do not require bilevel programming, so we analyze this additional complexity brought by calculating hyper-gradients in Alg. 1. Hyper-gradients calculation requires a time complexity of $O(cK)$ and space complexity of $O(n_2 K)$, where $c = c(n_1, n_2)$ is the time complexity to compute the dynamics Ξ and n_1, n_2 are the dimensions of the θ and ψ. For the sake of computational complexity, we use a truncated version of hyper-gradient as proposed by [44]. Specially, we truncate the computation and estimate the hyper-gradient every K' iterations where $K' < K$.

5 EXPERIMENTS

5.1 Datasets

To examine the performance of the proposed model, we report comparative experiment results from synthetic and real-world datasets. These datasets are carefully chosen from previous publications in a way to possess various ranges of properties, such as sequence length and number of event types. Sequences in each dataset are assumed to be independent and identically distributed. Table 1 summarizes statistical information of all datasets. The synthetic dataset is originally generated by [54] with the Python package *tick* [1]. Real-world datasets are taken from previous research works [2]. To facilitate the training and evaluation process of the models, each dataset consists of three subsets: training, validation and test. All reported values are from the test subsets. Following previous work, we compare models according to their testing loss and prediction performance [16, 39, 54].

5.2 Real-world Datasets

Retweet. The Retweet dataset contains information of when a post will be retweeted by which type of users. It contains a total number of 24,000 retweet sequences. In each sequence, an event is a tuple of the tweet type and time. There are $U = 3$ types: "small", "medium" and "large". The "small" retweeters are those who have fewer than 120 followers, "medium" retweeters have more than 120 but fewer than 1,363 followers, and the rest are "large" retweeters. The proportion of the "small", "medium" and "large" type is 50%,45% and 5% respectively. As for retweet time, the first event in each sequence is labeled with 0, while the next events are labeled with reference to their time interval with respect to the first event in this sequence.

StackOverflow. The StackOverflow dataset includes sequences of user awards within two years. StackOverflow is a question-answering website where users are awarded based on their proposed questions and answers to questions by others. This dataset contains in total 6,633 sequences. There are $U = 22$ types of events: Nice Question, Good Answer, Guru, Popular Question, Famous Question, Nice Answer, Good Question, Caucus, Notable Question, Necromancer, Promoter, Yearling, Revival, Enlightened, Great Answer, Populist, Great Question, Constituent, Announcer, Stellar

[1] https://github.com/X-DataInitiative/tick
[2] https://drive.google.com/drive/folders/0BwqmV0EcoUc8UklIR1BKV25YR1U

Table 1: Statistics of the used datasets.

Dataset	# of Types	Sequence Length			# of Sequences		
		Min	Mean	Max	Train	Validation	Test
Synthetic	2	68	132	269	3,200	400	400
Retweet	3	50	109	264	20,000	2,000	2,000
StackOverflow	22	41	72	736	4,777	530	1,326
MIMIC	75	2	4	33	527	58	65

Table 2: The KL-divergence of the estimated intensity against the ground truth.

Event type	RMTPP	CTLSTM	FullyNN	LogNormMix	SAHP	THP	Ours
1	1.89	1.75	1.55	1.53	1.46	1.45	**1.22** ± 0.06
2	1.72	1.69	1.47	1.47	1.24	1.23	**1.08** ± 0.08

Table 3: Negative log-likelihood per event on the four test sets.

Dataset	RMTPP	CTLSTM	FullyNN	LogNormMix	SAHP	THP	Ours
Synthetic	1.85	1.83	1.55	1.43	1.35	1.33	**1.17**±0.03
Retweet	7.43	6.95	6.23	5.32	4.56	4.57	**3.89**±0.08
StackOverflow	2.44	2.38	2.21	2.01	1.86	1.84	**1.54**±0.06
MIMIC	1.33	1.36	1.03	0.78	0.52	0.54	**0.43**±0.05

Question, Booster and Publicist. With this dataset, we learn which type of awards will be given to a user and when.

MIMIC-II. The Multiparameter Intelligent Monitoring in Intensive Care (MIMIC-II) dataset is developed based on an electric medical record system. The dataset contains 650 sequences, each of which corresponds to a patient's clinical visits in a seven-year period. Each clinical event records the diagnosis result and the timestamp of that visit. The number of unique diagnosis results is $U = 75$. According to the clinical history, a temporal point process is supposed to capture the dynamics of when a patient will visit doctors and what the diagnose result will be.

5.3 Experimental Setup

For a fair comparison, beside using the train, validation and test sets already defined by the dataset, we further split the training set into a sub-training set (80%) and a sub-validation set (20%), which are used in the lower and upper objective functions respectively. We tune the hyper-parameters based on the validation set and report performance on the test set. We explore the number of attention heads in the set $\{1, 2, 4, 8, 16\}$. Another hyper-parameter is the number of attention layers. We explore this hyper-parameter in the set $\{2, 3, 4, 5, 6\}$. The dependency graph is implemented as a mask, and we conduct a logical conjunction with the mask that prevents future attending. We use Adam to optimize both the lower and upper objectives. The learning rate η ranges from 10^{-7} to 10^{-4} with a multiplication step of 10 while the sparsity parameter ρ ranges from 0.3 to 0.9 with an addition step of 0.1. The scaling factor β is chosen from $[0.2, 0.4, 0.8, 1.6, 3.2, 6.4]$. We employ tricks including dropout, weight regularization and early stopping to prevent overfitting. Mini-batch size is set to be 32, which allowed a proper use of the memory of our Nvidia GeForce GTX 1080 cards.

5.4 Baselines

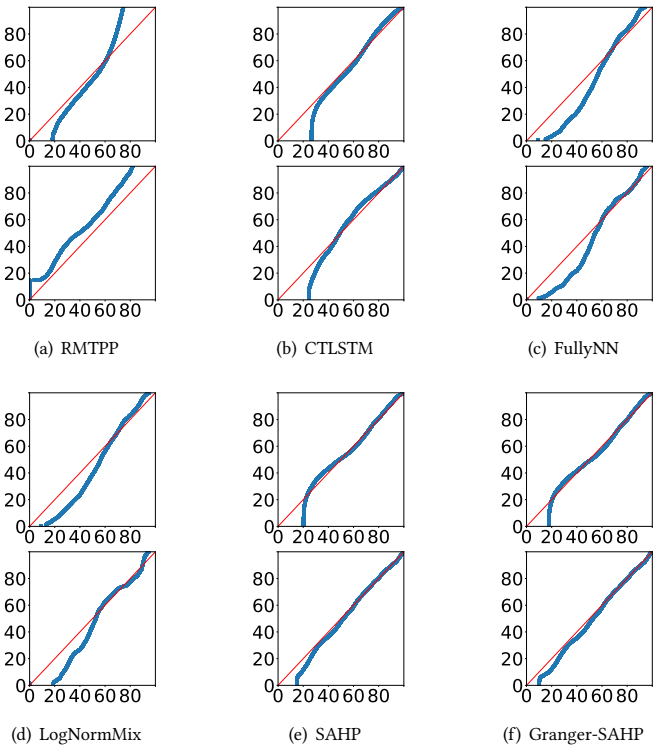
To evaluate our model, we compare it with state-of-the-art NPP models. All the baselines are trained as indicated in their papers.

- Recurrent Marked Temporal Point Processes (RMTPP). This model [16] uses RNN to learn a representation of influences from past events, and time intervals are explicitly encoded.
- Continuous Time LSTM (CTLSTM). [39] use a continuous-time LSTM, which includes intensity decay and eliminate the need to encode event intervals as numerical inputs.
- Fully Neural Network (FullyNN).[42] propose to model the cumulative intensity function with a feed-forward network.
- Log Normal Mixture (LogNormMix).[46] suggest to model the conditional probability density distribution by a log-normal mixture model.
- Self-Attentive Hawkes Process (SAHP) [54] uses self-attention to calculate intensity with time-shifted position embeddings.
- Transformer Hawkes Process (THP) [57]) is a concurrent self-attention based point process model with additional structural knowledge.

6 RESULTS AND DISCUSSION

For a fair comparison, we tried different hyper-parameter configurations for baselines and our model, and selected those with the best validation performance. Because the proposed model is a randomized procedure, we report both averages and standard deviations of our achieved performance.

Goodness of fit on the synthetic dataset. With the synthetic dataset, the true intensity is available and we are able to evaluate the goodness-of-fit of the estimated intensity. We compute the



(a) RMTPP (b) CTLSTM (c) FullyNN

(d) LogNormMix (e) SAHP (f) Granger-SAHP

Figure 2: QQ-plot of true VS estimated intensities of types. The x-axis and the y-axis represent the quantiles of the true and estimated intensities. For each model the top figure is for the type-1 events while the bottom figure is for the type-2 events.

KL-divergence $KL(p||q)$ between the true distribution and the estimated one, where p and q indicate the truth and the estimation respectively; results are shown in Table 2. We achieve the lowest KL-divergence, 1.22 and 1.08 on average, for the two event types, indicating that our model is able to best approximate the intensity. From the QQ-plots of estimated intensity in Figure 2, we observe that the intensity estimated by SAHP produces the most similar distribution to the true one, which indicates that SAHP is able to best capture the underlying complicated dynamics of the synthetic dataset.

Sequence Modelling. We evaluate the ability of the models using the negative log-likelihood (NLL) as evaluation metric, as done in prior works [39, 46]. The higher the NLL is the less a model is able to model an sequence of events. Tab. 3 presents the results of this evaluation on each test subset. We observe that that our model achieves the best average NLL loss with small standard deviations, outperforming the baselines in all datasets. RMTPP and CTLSTM have very similar performance except on the Retweet dataset, where CTLSTM achieves a lower NLL than RMTPP.

Event Dependency. This part interprets what the proposed model learns from observed event sequences after convergence. Taking the StackOverflow dataset as an example, we visualize in Fig. 3 the learnt parameter θ on the left and a sampled graph on the right. This parameter θ contains the Bernoulli distributions of edges between pairs of event types. The cell at the u-th row and the v-th column is the $\theta_{u,v}$ indicating the probability that type-v events contribute to predicting type-u events. In this figure we observe that the elements in the diagonal are all above 0.5, i.e., these event

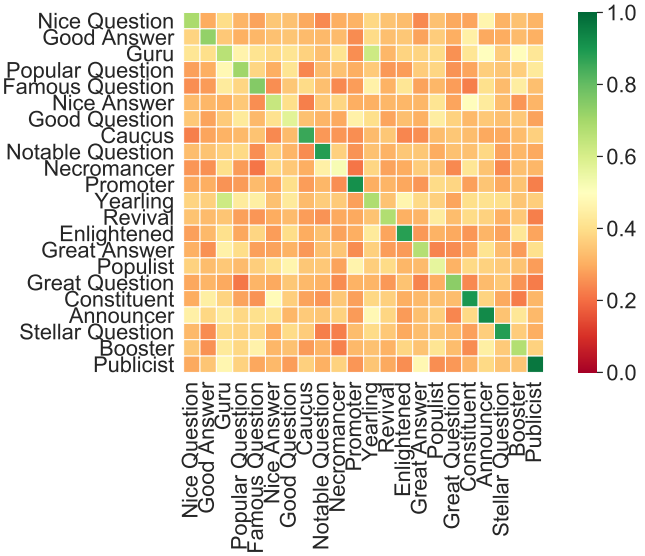


Figure 3: Visualization of the latent dependency graph (θ)

Table 4: $F_1(\%)$ and NRMSE for event prediction on the four test subsets.

Dataset	Synthetic		Retweet		StackOverflow		MIMIC	
	F_1	NRMSE	F_1	NRMSE	F_1	NRMSE	F_1	NRMSE
RMTPP	40.32	37.07	41.22	1276.41	5.44	207.79	28.76	6.83
CTLMST	43.80	35.08	39.21	1255.05	4.88	194.87	34.00	6.49
FullyNN	45.21	33.34	43.80	1104.41	6.34	173.92	33.32	5.43
LogNormMix	42.09	32.64	45.25	1090.45	3.23	154.13	32.86	4.12
SAHP	58.50	31.32	53.92	1055.05	24.12	133.61	36.90	3.89
THP	58.45	31.24	53.86	1054.43	23.89	134.02	37.05	3.76
Ours	**62.18**±0.23	**30.51**±0.52	**58.29**±0.63	**1021.47**±1.43	**28.34**±0.19	**114.74** ±3.24	**38.80**±0.81	**3.42**±0.66

Table 5: Model running time (Seconds) on the retweet dataset with a Titan Xp GPU card.

Model	RMTPP	CTLSTM	LogNormMix	FullyNN	SAHP	THP	Ours
Time	92.22	134.69	**85.32**	87.53	86.97	87.49	95.34

types own a self-excitation property. Some types predicts others and such dependency pattern can be sparse and asymmetric.

Event Prediction. We compare the prediction performance of our model against the state-of-the-art baselines. The task here is to predict the next event, including type and timestamp, given historical events. Event type prediction is a multi-class classification problem and we evaluate performance using micro F_1. Because a time interval is a real number, we use the Normalized Root Mean Square Error (NRMSE) as the evaluation metric, defined as $\varepsilon_i = (\hat{t}_{i+1} - t_{i+1})/(t_{i+1} - t_i)$ where $\hat{t}_{i+1}$ is the predicted timestamp and t_{i+1} is the ground truth, and $t_{i+1} - t_i$ is the ground-truth time interval. The results in Tab. 4 show that our model outperforms the baselines in terms of F_1 and NRMSE on all prediction tasks. One interesting finding is the incorporation of the latent graph leads to a clear improvement in event type prediction in terms of F_1.

Computational efficiency. To compare the computational efficiency of our method with neural baselines, we report in Table 5 the running time on the retweet dataset with a Titan Xp GPU card. The mini-batch size is 32 and running time is averaged by 10 epochs. We observe that our method spends longer time attention-based NPPs, although we employ an efficient implementation of bilevel programming. The average running time of our method is almost 10 seconds longer than SAHP and even longer than discrete RNN-based NPPs, but still less than continuous LSTM-based NPPs.

Graph Sparsity. Graph sparsity is a fundamental property that influences graph structures. In this paper, we use ρ to control the graph sparsity: the smaller ρ is, the sparser the graph is. To illustrate the influence of graph sparsity, we tune ρ and show the corresponding evaluation performance on the StackOverflow dataset. Table 6 describes how the performance of event prediction changes with different ρ. We report the average values only. The best performance is achieved at $\rho = 0.7$.

Table 6: The influence of ρ on the performance.

ρ	0.3	0.4	0.5	0.6	0.7	0.8	0.9
NLL	2.08	1.90	1.83	1.68	**1.54**	1.65	1.79
F_1	21.80	23.16	25.65	26.65	**28.34**	27.03	25.89
NRMSE	140.84	135.61	129.43	121.54	**114.74**	119.43	126.05

7 RELATED WORK

Neural Point Process. Point process conventionally assumes a fixed form of intensity, such as Hawkes process with an exponential kernel, to account for the additive influence of an history event. To increase the capacity of point process, neural networks have been adopted to consider more complex phenomenon such as preventive effects and non-exponential influence. Initial works used recurrent neural networks either in a discrete way [16], a continuous way [39], or a multi-scale multi-channel architecture [24]; then self-attention was also used by [54, 57]. [57] develop a Structured Transformer Hawkes (STH) model that studies the relationship between point processes. It assumes the modeling of one point process might contribute to the modeling of another. Each vertex in the STH's graph is associated with a point process. Other efforts have been made to model the cumulative intensity function [42] and conditional probability density [46]. One issue that has been ignored in these works is that these NPP models take all types of historic events to predict the future ones, neglecting that some event types may not contribute to the prediction of another type. This study aims to correct this defect by learning to omit those non-contributing event types when predicting the target type via NPPs.

Graphs have been incorporated into temporal point processes. Linderman and Adams [33] aims to discover the latent graph structure among individual events instead of event types. Linderman and Adams [33] use a probabilistic method to extract mutual influence among individual events rather than event types. Bhattacharjya et al. [6] propose a proximal graphical event model to reveal relationships among event types, but this model makes a parametric form of the intensity function and is not as capable as NPPs. Shang and Sun [45] incorporate graph convolutional networks to capture the correlation between event sequences into Hawkes processes but they use undirected graphs while our graph is directed. Wu et al. [50] model event propagation via graph biased NPPs; the graph is

a visible followship network among people on social media. Most of these works assume their graphs are already existing and they directly input the graph to a neural network while we generate a latent graph by a probabilistic graph generator.

Granger Causality for Point Processes. Granger causality can identify a subset of event types $\mathcal{V} \subseteq \mathcal{U}$ for the type-u event, such that $\lambda_u(t)$ only depends on historical events of types in $\mathcal{V}$ [11, 28, 38]. Eichler et al. [17] propose to connect Granger causality with $\phi_{u,v}$. Lasso and sparse-group-lasso have also been explored in [2, 52]. Achab et al. [1] uncover the Granger causality of Hawkes processes by learning the kernels' integrals. These methods do not employ neural networks thus may not capture complicated event dynamics. [48] estimate Granger causality by likelihood reduction while [55] use an axiomatic attribution method to convert event contributions to a GC statistic. These methods are performed after the fitting of the model (in post-processing) yet these NPP models still suffer from the same issue thereof, i.e., they use all event types. The latent graph we generate is quite similar to Granger causality, but the difference is that Granger causality is deterministic while the latent graph is probabilistic.

Uncovering the Latent Graph Structure. As this paper involves a latent graph, we review works on how to uncover latent graphs from observed data. This line of works are mainly in the areas of graph and network analysis. The task of link prediction infers the existence of an edge between a pair of vertices [35] yet link predictions are mostly semi-supervised or require vertex information. Graph generators based on deep learning are aimed to reflect properties of available graphs [27, 32, 53]. Johnson [29] propose a differential neural model to produce graphs but requiring supervision from ground truth. Kipf et al. [30] use an encoder-decoder to infer latent relationships between entities and capture the dynamics of physical systems. Franceschi et al. [23] employ a graph neural network to uncover graph structures and classify vertices.

8 CONCLUSION

Not all event types contribute to the prediction of one target type. This paper aims to remove disturbance of non-contributing types from NPPs. We simultaneously consider two tasks: (1) finding a set of contributing event types and (2) learning NPP from event sequences. We propose a probabilistic graph generator, from which a graph is sampled and then used to modify a NPP. As the tasks are nested, we formulate them by bilevel programming and develop an efficient solution. Using validation based log-likelihood that is due to choice of the graphical structure to tune edge-probabilities is a reasonable idea, and reflects the popular practice of using a validation set to tune parameters in the interest of generalization.

ACKNOWLEDGMENTS

This project was funded by the EPSRC Fellowship titled "Task Based Information Retrieval", grant reference number EP/P024289/1. We acknowledge the support of NVIDIA Corporation with the donation of the Titan Xp GPU.

REFERENCES

[1] Massil Achab, Emmanuel Bacry, Stéphane Gaïffas, Iacopo Mastromatteo, and Jean-François Muzy. 2017. Uncovering Causality from Multivariate Hawkes Integrated Cumulants. *J. Mach. Learn. Res.* 18, 1 (Jan. 2017).
[2] Amr Ahmed and Eric P Xing. 2009. Recovering time-varying networks of dependencies in social and biological studies. *Proceedings of the National Academy of Sciences* 106, 29 (2009), 11878–11883.
[3] Emmanuel Bacry and Jean-François Muzy. 2014. Hawkes model for price and trades high-frequency dynamics. *Quantitative Finance* 14, 7 (2014), 1147–1166.
[4] Yoshua Bengio, Nicholas Léonard, and Aaron Courville. 2013. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432* (2013).
[5] Kristin P Bennett, Jing Hu, Xiaoyun Ji, Gautam Kunapuli, and Jong-Shi Pang. 2006. Model selection via bilevel optimization. In *The 2006 IEEE International Joint Conference on Neural Network Proceedings*. IEEE, 1922–1929.
[6] Debarun Bhattacharjya, Dharmashankar Subramanian, and Tian Gao. 2018. Proximal graphical event models. In *Advances in Neural Information Processing Systems*. 8136–8145.
[7] B. Bollobás. 1985. *Random graphs*. Academic Press. https://books.google.co.uk/books?id=2uvuAAAAMAAJ
[8] Béla Bollobás and Bollobás Béla. 2001. *Random graphs*. Number 73. Cambridge university press.
[9] Léon Bottou. 2010. Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010*. Springer, 177–186.
[10] David R Brillinger, Peter M Guttorp, Frederic Paik Schoenberg, Abdel H El-Shaarawi, and Walter W Piegorsch. 2002. Point processes, temporal. *Encyclopedia of Environmetrics* 3 (2002), 1577–1581.
[11] Kacper Chwialkowski and Arthur Gretton. 2014. A kernel independence test for random processes. In *International Conference on Machine Learning*. 1422–1430.
[12] Benoît Colson, Patrice Marcotte, and Gilles Savard. 2007. An overview of bilevel optimization. *Annals of operations research* 153, 1 (2007), 235–256.
[13] David Roxbee Cox and Valerie Isham. 1980. *Point processes*. Vol. 12. CRC Press.
[14] Daryl J Daley and David Vere-Jones. 2007. *An introduction to the theory of point processes: volume II: general theory and structure*. Springer Science & Business Media.
[15] Justin Domke. 2012. Generic methods for optimization-based modeling. In *Artificial Intelligence and Statistics*. 318–326.
[16] Nan Du, Hanjun Dai, Rakshit Trivedi, Utkarsh Upadhyay, Manuel Gomez-Rodriguez, and Le Song. 2016. 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*. ACM, 1555–1564.
[17] Michael Eichler, Rainer Dahlhaus, and Johannes Dueck. 2017. Graphical modeling for multivariate hawkes processes with nonparametric link functions. *Journal of Time Series Analysis* 38, 2 (2017), 225–242.
[18] Paul Erdos. 1959. On random graphs. *Publicationes mathematicae* 6 (1959), 290–297.
[19] Mehrdad Farajtabar, Nan Du, Manuel Gomez Rodriguez, Isabel Valera, Hongyuan Zha, and Le Song. 2014. Shaping social activity by incentivizing users. In *Advances in neural information processing systems*. 2474–2482.
[20] Mehrdad Farajtabar, Yichen Wang, Manuel Gomez Rodriguez, Shuang Li, Hongyuan Zha, and Le Song. 2015. Coevolve: A joint point process model for information diffusion and network co-evolution. In *Advances in Neural Information Processing Systems*. 1954–1962.
[21] Rémi Flamary, Alain Rakotomamonjy, and Gilles Gasso. 2014. Learning constrained task similarities in graphregularized multi-task learning. *Regularization, Optimization, Kernels, and Support Vector Machines* (2014), 103.
[22] Luca Franceschi, Michele Donini, Paolo Frasconi, and Massimiliano Pontil. 2017. Forward and reverse gradient-based hyperparameter optimization. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, 1165–1173.
[23] Luca Franceschi, Mathias Niepert, Massimiliano Pontil, and Xiao He. 2019. Learning discrete structures for graph neural networks. *arXiv preprint arXiv:1903.11960* (2019).
[24] Tian Gao, Dharmashankar Subramanian, Karthikeyan Shanmugam, Debarun Bhattacharjya, and Nicholas Mattei. 2020. A Multi-Channel Neural Graphical Event Model with Negative Evidence.. In *AAAI*. 3946–3953.
[25] Edgar N Gilbert. 1959. Random graphs. *The Annals of Mathematical Statistics* 30, 4 (1959), 1141–1144.
[26] Andreas Griewank and Andrea Walther. 2008. *Evaluating derivatives: principles and techniques of algorithmic differentiation*. Vol. 105. Siam.
[27] Aditya Grover, Aaron Zweig, and Stefano Ermon. 2018. Graphite: Iterative generative modeling of graphs. *arXiv preprint arXiv:1803.10459* (2018).
[28] Asela Gunawardana, Christopher Meek, and Puyang Xu. 2011. A model for temporal dependencies in event streams. In *Advances in Neural Information Processing Systems*. 1962–1970.
[29] Daniel D Johnson. 2016. Learning graphical state transitions. (2016).

[30] Thomas Kipf, Ethan Fetaya, Kuan-Chieh Wang, Max Welling, and Richard Zemel. 2018. Neural relational inference for interacting systems. *arXiv preprint arXiv:1802.04687* (2018).

[31] Shuang Li, Shuai Xiao, Shixiang Zhu, Nan Du, Yao Xie, and Le Song. 2018. Learning temporal point processes via reinforcement learning. In *Advances in neural information processing systems*. 10781–10791.

[32] Yujia Li, Oriol Vinyals, Chris Dyer, Razvan Pascanu, and Peter Battaglia. 2018. Learning deep generative models of graphs. *arXiv preprint arXiv:1803.03324* (2018).

[33] Scott Linderman and Ryan Adams. 2014. Discovering latent network structure in point process data. In *International Conference on Machine Learning*. 1413–1421.

[34] James Lloyd, Peter Orbanz, Zoubin Ghahramani, and Daniel M Roy. 2012. Random function priors for exchangeable arrays with applications to graphs and relational data. In *Advances in Neural Information Processing Systems*. 998–1006.

[35] Linyuan Lü and Tao Zhou. 2011. Link prediction in complex networks: A survey. *Physica A: statistical mechanics and its applications* 390, 6 (2011), 1150–1170.

[36] Dougal Maclaurin, David Duvenaud, and Ryan Adams. 2015. Gradient-based hyperparameter optimization through reversible learning. In *International Conference on Machine Learning*. 2113–2122.

[37] Chris J Maddison, Andriy Mnih, and Yee Whye Teh. 2016. The concrete distribution: A continuous relaxation of discrete random variables. *arXiv preprint arXiv:1611.00712* (2016).

[38] Christopher Meek. 2014. Toward learning graphical and causal process models. In *Proceedings of the UAI 2014 Conference on Causal Inference: Learning and Prediction-Volume 1274*. CEUR-WS. org, 43–48.

[39] Hongyuan Mei and Jason M Eisner. 2017. The neural hawkes process: A neurally self-modulating multivariate point process. In *Advances in Neural Information Processing Systems*. 6754–6764.

[40] Luis Muñoz-González, Battista Biggio, Ambra Demontis, Andrea Paudice, Vasin Wongrassamee, Emil C Lupu, and Fabio Roli. 2017. Towards poisoning of deep learning algorithms with back-gradient optimization. In *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security*. 27–38.

[41] Yosihiko Ogata. 1981. On Lewis' simulation method for point processes. *IEEE Transactions on Information Theory* 27, 1 (1981), 23–31.

[42] Takahiro Omi, Naonori Ueda, and Kazuyuki Aihara. 2019. Fully Neural Network based Model for General Temporal Point Processes. *arXiv preprint arXiv:1905.09690* (2019).

[43] Jakob Gulddahl Rasmussen. 2018. Lecture Notes: Temporal Point Processes and the Conditional Intensity Function. *arXiv preprint arXiv:1806.00221* (2018).

[44] Amirreza Shaban, Ching-An Cheng, Nathan Hatch, and Byron Boots. 2018. Truncated back-propagation for bilevel optimization. *arXiv preprint arXiv:1810.10667* (2018).

[45] Jin Shang and Mingxuan Sun. 2019. Geometric Hawkes Processes with Graph Convolutional Recurrent Neural Networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 33. 4878–4885.

[46] Oleksandr Shchur, Marin Biloš, and Stephan Günnemann. 2019. Intensity-Free Learning of Temporal Point Processes. *arXiv preprint arXiv:1909.12127* (2019).

[47] H Von Stackelberg. 1952. The theory of market economy Oxford university Press.

[48] Tianyu Wang, Christian Walder, and Tom Gedeon. 2018. Neural Causality Detection for Multi-dimensional Point Processes. In *International Conference on Neural Information Processing*. Springer, 509–521.

[49] Yichen Wang, Bo Xie, Nan Du, and Le Song. 2016. Isotonic hawkes processes. In *International conference on machine learning*. 2226–2234.

[50] Weichang Wu, Huanxi Liu, Xiaohu Zhang, Yu Liu, and Hongyuan Zha. 2020. Modeling Event Propagation via Graph Biased Temporal Point Process. *IEEE Transactions on Neural Networks and Learning Systems* (2020).

[51] Shuai Xiao, Mehrdad Farajtabar, Xiaojing Ye, Junchi Yan, Le Song, and Hongyuan Zha. 2017. Wasserstein learning of deep generative point process models. In *Advances in Neural Information Processing Systems*. 3247–3257.

[52] Hongteng Xu, Mehrdad Farajtabar, and Hongyuan Zha. 2016. Learning granger causality for hawkes processes. In *International Conference on Machine Learning*. 1717–1726.

[53] Jiaxuan You, Rex Ying, Xiang Ren, William L Hamilton, and Jure Leskovec. 2018. Graphrnn: A deep generative model for graphs. *arXiv preprint arXiv:1802.08773* (2018).

[54] Qiang Zhang, Aldo Lipani, Omer Kirnap, and Emine Yilmaz. 2019. Self-Attentive Hawkes Processes. arXiv:1907.07561 [cs.LG]

[55] Wei Zhang, Thomas Kobber Panum, Somesh Jha, Prasad Chalasani, and David Page. 2020. CAUSE: Learning Granger Causality from Event Sequences using Attribution Methods. *arXiv preprint arXiv:2002.07906* (2020).

[56] Shengyu Zhu, Ignavier Ng, and Zhitang Chen. 2019. Causal discovery with reinforcement learning. *arXiv preprint arXiv:1906.04477* (2019).

[57] Simiao Zuo, Haoming Jiang, Zichong Li, Tuo Zhao, and Hongyuan Zha. 2020. Transformer Hawkes Process. *arXiv preprint arXiv:2002.09291* (2020).

Appendix

A MODIFY THE INTENSITY FUNCTION WITH THE SAMPLED GRAPH

We choose to modify the self-attention based neural point process [54] because: (1) the dependency factor $\mathcal{G}_{u,v}$ can be easily used as a plug-in to remove certain types of events from history if these types do not contribute to the prediction of another event type, and (2) it is a state-of-the-art point process model. Specifically, we modify the conventional way of computing attention scores, i.e., embedded Gaussian, by multiplying $\mathcal{G}_{u,v} \in \{0, 1\}$,

$$f(\boldsymbol{x}_{i+1}, \boldsymbol{x}_j) = \mathcal{G}_{u,v} \cdot \exp\left(\boldsymbol{x}_{i+1}\boldsymbol{x}_j^T\right), \tag{15}$$

where u and v are the types of the $i+1$-th and j-th event. $\mathcal{G}_{u,v}$ is a plug-in, which hinders historical influence of type-v events if type-v does not contribute to predicting type-u.

The rest of how to compute the intensity is the same with [54]: (a) the attention scores $f(\cdot, \cdot)$ is used to calculate a hidden embedding $\boldsymbol{h}_{u,i+1}$ that summarizes influence of history events before t_{i+1} on the type-u event at the $(i+1)$-th position of a sequence,

$$\boldsymbol{h}_{u,i+1} = \left(\sum_{j=1}^{i} f(\boldsymbol{x}_{i+1}, \boldsymbol{x}_j) g(\boldsymbol{x}_j)\right) / \sum_{j=1}^{i} f(\boldsymbol{x}_{i+1}, \boldsymbol{x}_j), \tag{16}$$

where the function $g(\cdot)$ is a linear transformation; (b) $\boldsymbol{h}_{u,i+1}$ is used to calculate three dynamic parameters $\mu_{u,i+1}$, $\eta_{u,i+1}$ and $\gamma_{u,i+1}$,

$$\mu_{u,i+1} = \text{gelu}\left(\boldsymbol{h}_{u,i+1} W_\mu\right), \tag{17}$$

$$\eta_{u,i+1} = \text{gelu}\left(\boldsymbol{h}_{u,i+1} W_\eta\right), \tag{18}$$

$$\gamma_{u,i+1} = \text{softplus}\left(\boldsymbol{h}_{u,i+1} W_\gamma\right); \tag{19}$$

and (c) the three parameters are used to compute the intensity function,

$$\lambda_u(t) = \text{softplus}(\mu_{u,i+1} + (\eta_{u,i+1} - \mu_{u,i+1}) \exp(-\gamma_{u,i+1}(t - t_i))). \tag{20}$$

B EXPERIMENTS

B.1 Synthetic Dataset

We generate a synthetic dataset using the open-source Python library *tick*. A two-dimensional point process is generated with base intensities $\mu_1 = 0.1$ and $\mu_2 = 0.2$. The triggering kernels consist of a power law kernel, an exponential kernel, a sum of two exponential kernels, and a sine kernel:

$$\phi_{1,1}(t) = 0.2 \times (0.5 + t)^{-1.3} \tag{21a}$$

$$\phi_{1,2}(t) = 0.03 \times \exp(-0.3t) \tag{21b}$$

$$\phi_{2,1}(t) = 0.05 \times \exp(-0.2t) + 0.16 \times \exp(-0.8t) \tag{21c}$$

$$\phi_{2,2}(t) = \max(0, \sin(t)/8) \quad \text{for } 0 \le t \le 4 \tag{21d}$$

In Figure 4 we show the four triggering kernels of the 2-dimensional point processes. Simulation is conducted based on Ogata's simulation algorithm [41].

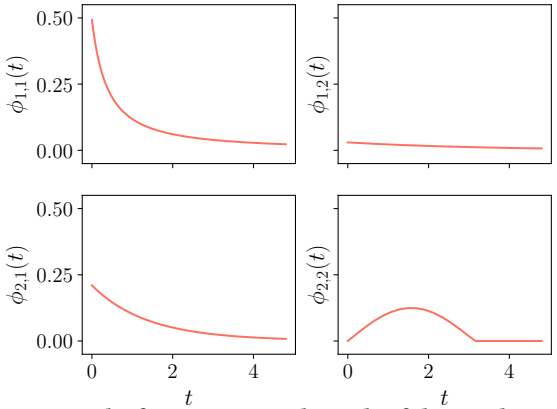


Figure 4: The four triggering kernels of the synthetic dataset with 2 event types.