Transformer Hawkes Process

Simiao Zuo*
Georgia Tech
Atlanta, USA
simiaozuo@gatech.edu

Haoming Jiang
Georgia Tech
Atlanta, USA
jianghm@gatech.edu

Zichong Li
USTC
Heifei, China
lzc123@mail.ustc.edu.cn

Tuo Zhao*
Georgia Tech
Atlanta, USA
tourzhao@gatech.edu

Hongyuan Zha*†
CUHK, Shenzhen
Shenzhen Institute of Artificial Intelligence and Robotics for Society
Shenzhen, China
zhahy@cuhk.edu.cn

ABSTRACT

Modern data acquisition routinely produce massive amounts of event sequence data in various domains, such as social media, healthcare, and financial markets. These data often exhibit complicated short-term and long-term temporal dependencies. However, most of the existing recurrent neural network based point process models fail to capture such dependencies, and yield unreliable prediction performance. To address this issue, we propose a Transformer Hawkes Process (THP) model, which leverages the self-attention mechanism to capture long-term dependencies and meanwhile enjoys computational efficiency. Numerical experiments on various datasets show that THP outperforms existing models in terms of both likelihood and event prediction accuracy by a notable margin. Moreover, THP is quite general and can incorporate additional structural knowledge. We provide a concrete example, where THP achieves improved prediction performance for learning multiple point processes when incorporating their relational information.

CCS CONCEPTS

• **Computing methodologies** → *Neural networks.*

KEYWORDS

Computational social science, Temporal point process

ACM Reference Format:
Simiao Zuo, Haoming Jiang, Zichong Li, Tuo Zhao, and Hongyuan Zha. 2020. Transformer Hawkes Process. In *Proceedings of 1st ACM SIGKDD Workshop on Deep Learning for Spatiotemporal Data, Applications, and Systems (KDD '20 DeepSpatial Workshop).* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

*Corresponding author.
†Currently on leave from College of Computing, Georgia Institute of Technology.

1 INTRODUCTION

Event sequence data are naturally observed in our daily life. Through social media such as Twitter and Facebook, we share our experiences and respond to other users' information [34]. In these websites, each user has a sequence of events such as tweets and interactions. Hundreds of millions of users generate large amounts of tweets, which are essentially sequences of events at different time stamps. Besides social media, event data also exist in domains like financial transactions [2] and personalized healthcare [31]. For example, in electronic medical records, tests and diagnoses of each patient can be treated as a sequence of events. Unlike other sequential data such as time series, event sequences tend to be asynchronous [26], which means time intervals between events are just as important as the order of them to describe their dynamics. Also, depending on specific application requirements, event data show sophisticated dependencies on their history.

Point process is a powerful tool for modeling sequences of discrete events in continuous time, and the technique has been widely applied. Hawkes process [11, 15] and Poisson point process are traditionally used as examples of point processes. However, the simplified assumptions of the complicated dynamics of point processes limit the models' practicality. As an example, Hawkes process states that all past events should have positive influences on the occurrence of current event. However, a user on Twitter may initiate tweets on different topics, and these events should be considered as unrelated instead of mutually-excited.

To alleviate the over-simplifications, likelihood-free methods [19, 32] and non-parametric models like kernel methods and splines [30] have been proposed, but the increasing complexity and quantity of collected data crave for more powerful models. With the development of neural networks, in particular deep neural networks, focuses have been placed on incorporating these flexible models into classical point processes. Because of the sequential nature of event steams, existing methods rely heavily on Recurrent Neural Networks (RNNs). Neural networks are known for their ability to capture complicated high-level features, in particular, RNNs have the representation power to model the dynamics of event sequence data. In previous works, either vanilla RNN [8] or its variants [21, 33] have been used and significant progress in terms of likelihood and event prediction have been achieved.

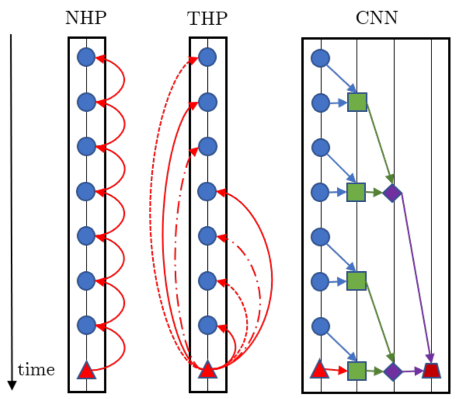

Figure 1: Illustration of dependency computation between the last event (the red triangle) and its history (the blue circles). RNN-based NHP can only model dependencies through recursion. THP directly and adaptively models event's dependencies on its history. Convolution-based models enforce static dependency patterns.

However, there are two significant drawbacks with RNN-based models. First, recurrent neural networks, even those equipped with forget gates, such as Long Short-Term Memory [14] and Gated Recurrent Units [6], are unlikely to capture long-term dependencies. In financial transactions, short-term effects such as policy changes are important for modeling buy-sell behaviors of stocks. On the other hand, because of the delays in asset returns, stock transactions and prices often exhibit long-term dependencies on their history. As another example, in medical domains, at times we are interested in examining short-term dependencies on symptoms such as fever and cough for acute diseases like pneumonia. But for certain types of chronic diseases such as diabetes, long-term dependencies on disease diagnoses and medications are more critical. Desirable models should be able to capture these long-term dependencies. Yet with recurrent structures, interactions between two events located far in the temporal domain are always weak [13], even though in reality they may be highly correlated. The reason is that the probability of keeping information in a state that is far away from the current state decreases exponentially with distance.

The second drawback is trainability of recurrent neural networks. Training deep RNNs (including LSTMs) is notoriously difficult because of gradient explosion and gradient vanishing [23]. In practice, single-layer and two-layer RNNs are mostly used, and they may not successfully model sophisticated dependencies among data [4]. Additionally, inputs are fed into the recurrent models sequentially, which means future states must be processed after the current state, rendering it impossible to process all the events in parallel. This limits RNNs' ability to scale to large problems.

Recently, convolutional neural network variants that are tailored for analyzing sequential data [10, 22, 35] have been proposed to better capture long-term effects. However, these models enforce many unnecessary dependencies. This particular downside plus the increased computational burdens deem these models insufficient.

To address the above concerns, we propose a Transformer Hawkes Process (THP) model that is able to capture both short-term and long-term dependencies whilst enjoying computational efficiency. Even though the Transformer [29] is widely adopted in natural language processing, it has rarely been used in other applications. We remark that such an architecture is not readily applicable to event sequences that are defined in a continuous-time domain. To the best of our knowledge, our proposed THP is the first of this type in point process literature.

Building blocks of THP are the self attention modules [3]. These modules directly model dependencies among events by assigning attention scores. A large score between two events implies a strong dependency, and a small score implies a weak one. In this way, the modules are able to adaptively select events that are at any temporal distance from the current event. Therefore, THP has the ability to capture both short-term and long-term dependencies. Figure 1 demonstrates dependency computation of different models.

The non-recurrent structure of THP facilitates efficient training of multi-layer models. Transformer-based architectures can be as deep as dozens of layers [7, 24], where deeper layers capture higher order dependencies. The ability to capture such dependencies creates models that are more powerful than RNNs, which are often shallow. Also, THP allows full parallelism when calculating dependencies across all events, i.e., the computation between any two event pairs is independent with each other. This yields a model presenting strong efficiency.

Our proposed model is quite general, and can incorporate additional structural knowledge to learn more complicated event sequence data, such as multiple point processes over a graph. In social networks, each user has her own sequence of events, like tweets and comments. Sequences among users can be related, for example, a tweet from a user may trigger retweets from her followers. We can use graphs to model these follower-followee relationships [9, 38], where each vertex corresponds to a specific user and each edge represents connections between the two associated users. We propose an extension to THP that integrates these relational graphs [5, 20] into the self-attention module via a similarity metric among users. Such a metric can be learned by our proposed graph regularization.

We experiment THP on five datasets to evaluate both validation likelihood and event prediction accuracy. Our THP model exhibits superior performance to RNN-based models in all these experiments. We further test our structured-THP on two additional datasets, where the model achieves improved prediction performance for learning multiple point processes when incorporating their relational information.

2 BACKGROUND

We briefly review Neural Hawkes Process [21] and Transformer [29] in this section.

Neural Hawkes Process generalizes the classical Hawkes process by parameterizing its intensity function with recurrent neural networks. Specifically,

$$\lambda(t) = \sum_{k=1}^{K} \lambda_k(t) = \sum_{k=1}^{K} f_k\left(\mathbf{w}_k^\top \mathbf{h}(t)\right), \; t \in (0, T],$$
$$\mathbb{P}[k_t = k] = \frac{\lambda_k(t)}{\lambda(t)},$$

where $\lambda(t)$ is the intensity function, K is the number of event types, and $\mathbf{h}(t)$s are the hidden states of the event sequence, obtained by a

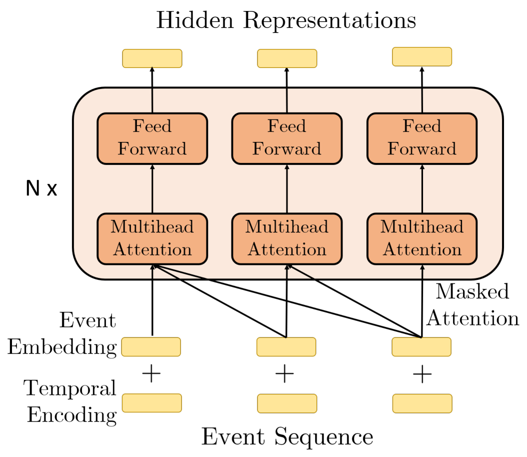

Figure 2: Architecture of the Transformer Hawkes Process. Each event sequence $\mathcal{S}$ is fed through embedding layers and N multi-head self-attention modules. Outputs of the THP are hidden representations of events in $\mathcal{S}$, with history information encoded.

continuous-time LSTM (CLSTM) module. CLSTM is an interpolated version of the standard LSTM, and it allows us to generate outputs in a continuous-time domain. Also, $f_k(x) = \beta_k \log\left(1 + \exp(x/\beta_k)\right)$ is the softplus function with parameter β_k that guarantees a positive intensity. One downside of the neural Hawkes process is that intrinsic weaknesses of RNNs are still inherited, namely the model is unable to capture long-term dependencies and is difficult to train.

Transformer is an attention-based model that has been broadly applied in tasks such as machine translation [7] and language modeling [24]. Despite its success in natural language processing, it has rarely been used in other areas. We remark that the Transformer architecture is not directly applicable to model point processes. In particular, time intervals between any two events can be arbitrary in event streams, while in natural languages, words are observed on regularly spaced time intervals. Therefore, we need to generalize the architecture to a continuous-time domain.

3 MODEL

We introduce our proposed Transformer Hawkes Process. Suppose we are given an event sequence $\mathcal{S} = \{(t_j, k_j)\}_{j=1}^{L}$ of L events, where each event has type $k_j \in \{1, 2, \dots, K\}$, with a total number of K types. Then each pair (t_j, k_j) corresponds to an event of type k_j occurs at time t_j.

3.1 Transformer Hawkes Process

The key ingredient of our proposed THP model is the self-attention module. Different from RNNs, the attention mechanism discards recurrent structures. However, our model still needs to be aware of the temporal information of inputs, i.e., time stamps. Therefore, analogous to the original positional encoding method [29], we propose to use a temporal encoding procedure, defined by

$$[\mathbf{z}(t_j)]_i = \begin{cases} \cos\left(t_j / 10000^{\frac{i-1}{M}}\right), & \text{if } i \text{ is odd}, \\ \sin\left(t_j / 10000^{\frac{i}{M}}\right), & \text{if } i \text{ is even}. \end{cases} \tag{1}$$

Eq. 1 uses trigonometric functions to define a temporal encoding for each time stamp, i.e., for each t_j, we deterministically computes $\mathbf{z}(t_j) \in \mathbb{R}^M$, where M is the dimension of encoding. Other temporal encoding methods can also be applied, such as the relative position representation model [27], where two temporal encoding matrices are learned instead of pre-defined.

Besides temporal encoding, we train an embedding matrix $\mathbf{U} \in \mathbb{R}^{M \times K}$ for the event types, where the k-th column of $\mathbf{U}$ is a M-dimensional embedding for event type k. For any event of type k_j, let $\mathbf{k}_j$ be its one-hot encoding (a K-dimensional vector with all 0s except for the k_j-th index, which has value 1), then its embedding is $\mathbf{U}\mathbf{k}_j$. Notice that for any event and its corresponding time stamp (t_j, k_j), the temporal encoding $\mathbf{z}(t_j)$ and the event embedding $\mathbf{U}\mathbf{k}_j$ both reside in $\mathbb{R}^M$. Embedding of the event sequence $\mathcal{S} = \{(t_j, k_j)\}_{j=1}^{L}$ is then specified by

$$\mathbf{X} = \left(\mathbf{U}\mathbf{Y} + \mathbf{Z}\right)^\top, \tag{2}$$

where $\mathbf{Y} = [\mathbf{k}_1, \mathbf{k}_2, \dots, \mathbf{k}_L] \in \mathbb{R}^{K \times L}$ is the collection of event type embedding, and $\mathbf{Z} = [\mathbf{z}(t_1), \mathbf{z}(t_2), \dots, \mathbf{z}(t_L)] \in \mathbb{R}^{M \times L}$ is the concatenation of event time encodings. Notice that $\mathbf{X} \in \mathbb{R}^{L \times M}$ and each row of $\mathbf{X}$ corresponds to the embedding of a specific event in the sequence.

After the initial encoding and embedding layers, we pass $\mathbf{X}$ through the self-attention module. Specifically we compute attention output $\mathbf{S}$ by

$$\begin{aligned} &\mathbf{S} = \text{Softmax}\left(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{M_K}}\right)\mathbf{V}, \\ &\mathbf{Q} = \mathbf{X}\mathbf{W}^Q, \quad \mathbf{K} = \mathbf{X}\mathbf{W}^K, \quad \mathbf{V} = \mathbf{X}\mathbf{W}^V. \end{aligned} \tag{3}$$

Here $\mathbf{Q}$, $\mathbf{K}$, and $\mathbf{V}$ are the query, key, and value matrices obtained by different transformations of $\mathbf{X}$, and $\mathbf{W}^Q, \mathbf{W}^K \in \mathbb{R}^{M \times M_K}$, $\mathbf{W}^V \in \mathbb{R}^{M \times M_V}$ are weights for the linear transformations, respectively. In practice using multi-head self-attention to increase model flexibility is more beneficial for data fitting. To facilitate this, different attention outputs $\mathbf{S}_1, \mathbf{S}_2, \dots, \mathbf{S}_H$ are computed using different sets of weights $\{W_j^Q, W_j^K, W_j^V\}_{j=1}^{H}$. The final attention output for the event sequence is then

$$\mathbf{S} = \left[\mathbf{S}_1, \mathbf{S}_2, \dots, \mathbf{S}_H\right]\mathbf{W}^O,$$

where $\mathbf{W}^O \in \mathbb{R}^{HM_V \times M}$ is an aggregation matrix.

We highlight that the self-attention module is able to directly select events whose occurrence time is at any distance from the current time. The j-th column of the attention weights $\text{Softmax}(\mathbf{Q}\mathbf{K}^\top / \sqrt{M_K})$ signifies event t_j's extent of dependency on its history. In contrast, RNN-based models encode history information sequentially via hidden representations of the events, i.e., the state of t_j depends on that of t_{j-1}, which in turn depends on t_{j-2}, etc. Should any of these encodings be weak, i.e., the RNN fails to learn sufficient relevant information for event t_k, hidden representations of any event t_j where $j \geq k$ will be inferior.

The attention output $\mathbf{S}$ is then fed through a position-wise feed-forward neural network, generating hidden representations $\mathbf{h}(t)$ of the input event sequence:

$$\begin{aligned} &\mathbf{H} = \text{ReLU}\left(\mathbf{S}\mathbf{W}_1^{\text{FC}} + \mathbf{b}_1\right)\mathbf{W}_2^{\text{FC}} + \mathbf{b}_2, \\ &\mathbf{h}(t_j) = \mathbf{H}(j, :). \end{aligned} \tag{4}$$

Here $\mathbf{W}_1^{\text{FC}} \in \mathbb{R}^{M \times M_H}$, $\mathbf{W}_2^{\text{FC}} \in \mathbb{R}^{M_H \times M}$, $\mathbf{b}_1 \in \mathbb{R}^{M_H}$, and $\mathbf{b}_2 \in \mathbb{R}^M$ are parameters of the neural network, and $\mathbf{W}_2^{\text{FC}}$ has identical columns.

The resulting matrix $\mathbf{H} \in \mathbb{R}^{L\times M}$ contains hidden representations of all the events in the input sequence, where each row corresponds to a particular event.

To avoid "peeking into the future", our attention algorithm is equipped with masks. That is, when computing the attention output $\mathbf{S}_j$ (the j-th column of $\mathbf{S}$), we mask all the future positions, i.e., we set $\mathbf{Q}_{j+1}, \mathbf{Q}_{j+2}, \ldots, \mathbf{Q}_L$ to 0. This will avoid the softmax function from assigning dependency to events in the future.

In practice we stack multiple self-attention modules together, and inputs are passed through each of these modules sequentially. In this way our model is able to capture high level dependencies. We remark that stacking RNN/LSTM is not plausible because gradient explosion and gradient vanishing will render the stacked model difficult to train. Figure 2 illustrates the architecture of THP.

3.2 Continuous Time Conditional Intensity

Dynamics of temporal point processes are described by a continuous conditional intensity function. Eq. 4 only generates hidden representations for discrete time stamps, and the associated intensity is also discrete. Therefore an interpolated continuous time intensity function is in need.

Let $\lambda(t|\mathcal{H}_t)$ be the conditional intensity function for our model, where $\mathcal{H}_t = \{(t_j, k_j)\}$ for all j such that $t_j < t$ is the history up to time t. We define different intensity functions for different event types, i.e., for every $k \in \{1, 2, \ldots, K\}$, define $\lambda_k(t|\mathcal{H}_t)$ as the conditional intensity function for events of type k. The conditional intensity function for the entire event sequence is defined by

$$\lambda(t|\mathcal{H}_t) = \sum_{k=1}^{K} \lambda_k(t|\mathcal{H}_t),$$

where each of the type-specific intensity takes the form

$$\lambda_k(t|\mathcal{H}_t) = f_k\Big(\underbrace{\alpha_k \frac{t-t_j}{t_j}}_{\textit{current}} + \underbrace{\mathbf{w}_k^\top \mathbf{h}(t)}_{\textit{history}} + \underbrace{b_k}_{\textit{base}}\Big). \tag{5}$$

In Eq. 5, time is defined on interval $t \in [t_j, t_{j+1})$, and $f_k(x) = \beta_k \log\left(1 + \exp(x/\beta_k)\right)$ is the softplus function with "softness" parameter β_k. The reason for choosing this particular function is two-fold: first, the softplus function ensures that the intensity is positive; second, "softness" of the softplus function guarantees stable computation and avoids dramatic changes in intensity.

Now we explain each term in Eq. 5 in detail:

⋄ The "*current*" influence is an interpolation between two observed time stamps t_j and t_{j+1}, and α_k modulates importance of the interpolation. When $t = t_j$, i.e., a new observation comes in, this influence is 0. When $t \to t_{j+1}$, the conditional intensity function is no longer continuous. As a matter of fact, Eq. 5 is continuous everywhere except for the observed events $\{(t_j, k_j)\}$. However, these "jumps" in intensity is a non-factor when computing likelihood.

⋄ The "*history*" term contains two parts: a vector $\mathbf{w}_k$ that transforms the hidden states of the THP model into a scalar, and the hidden states $\mathbf{h}(t)$ (Sec. 3.1) themselves that encode past events up to time t.

⋄ The "*base*" intensity represents probability of occurrence of events without considering history information.

With our proposed conditional intensity function, next time stamp prediction and next event type prediction is given by[1]

$$\begin{aligned} p(t|\mathcal{H}_t) &= \lambda(t|\mathcal{H}_t)\exp\Big(-\int_{t_j}^{t} \lambda(\tau|\mathcal{H}_\tau)d\tau\Big), \\ \widehat{t}_{j+1} &= \int_{t_j}^{\infty} t\cdot p(t|\mathcal{H}_t)dt, \\ \widehat{k}_{j+1} &= \operatorname*{argmax}_k \frac{\lambda_k(t_{j+1}|\mathcal{H}_{j+1})}{\lambda(t_{j+1}|\mathcal{H}_{j+1})}. \end{aligned} \tag{6}$$

3.3 Training

For any sequence $\mathcal{S}$ over an observation interval $[t_1, t_L]$, given its conditional intensity function $\lambda(t|\mathcal{H}_t)$, the log-likelihood is

$$\ell(\mathcal{S}) = \underbrace{\sum_{j=1}^{L} \log \lambda(t_j|\mathcal{H}_j)}_{\text{event log-likelihood}} - \underbrace{\int_{t_1}^{t_L} \lambda(t|\mathcal{H}_t)dt}_{\text{non-event log-likelihood}}. \tag{7}$$

Model parameters are learned by maximizing the log-likelihood across all sequences. Concretely, suppose we have N sequences $\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_N$, then the goal is to find parameters that solve

$$\max \ \textstyle\sum_{i=1}^{N} \ell(\mathcal{S}_i),$$

where $\ell(\mathcal{S}_i)$ is the log-likelihood of event sequence $\mathcal{S}_i$. This optimization problem can be efficiently solved by stochastic gradient type algorithms like ADAM [17]. Additionally, techniques that help stabilizing training such as layer normalization [1] and residual connection [12] are also applied.

In Eq. 7, one challenge is to compute $\Lambda = \int_{t_1}^{t_L} \lambda(t|\mathcal{H}_t)dt$, the non-event log-likelihood. Because of the softplus function, there is no closed-form computation for this integral, and a proper approximation is needed.

The first approach to approximate the non-event log-likelihood is by using Monte Carlo integration [25]:

$$\begin{aligned} \widehat{\Lambda}_{\text{MC}} &= \sum_{j=2}^{L}(t_j - t_{j-1})\Big(\frac{1}{N}\sum_{i=1}^{N}\lambda(u_i)\Big), \\ \nabla\widehat{\Lambda}_{\text{MC}} &= \sum_{j=2}^{L}(t_j - t_{j-1})\Big(\frac{1}{N}\sum_{i=1}^{N}\nabla\lambda(u_i)\Big). \end{aligned} \tag{8}$$

Here $u_i \sim \text{Unif}(t_{j-1}, t_j)$ is sampled from a uniform distribution with support $[t_{j-1}, t_j]$. Notice that $\lambda(u_i)$ and $\nabla\lambda(u_i)$ can be calculated by feed-forward and back-propagation through the model, respectively. Moreover, Eq. 8 yields an unbiased estimation to the integral, i.e., $\mathbb{E}[\widehat{\Lambda}_{\text{MC}}] = \Lambda$.

The second approach is to apply numerical integration methods, which are faster because of the elimination of sampling. For example, the trapezoidal rule [28] states that

$$\widehat{\Lambda}_{\text{NU}} = \sum_{j=2}^{L} \frac{t_j - t_{j-1}}{2}\Big(\lambda(t_j|\mathcal{H}_j) + \lambda(t_{j-1}|\mathcal{H}_{j-1})\Big) \tag{9}$$

qualifies as an approximation to Λ. Other higher order methods such as the Simpson's rule [28] can also be applied. Even though approximations build upon numerical integration algorithms are biased, in practice they are affordable. This is because the conditional intensity (Eq. 5) uses softplus as its activation function, which is

[1]Without causing any confusion, denote $\mathcal{H}_{t_j}$ as $\mathcal{H}_j$.

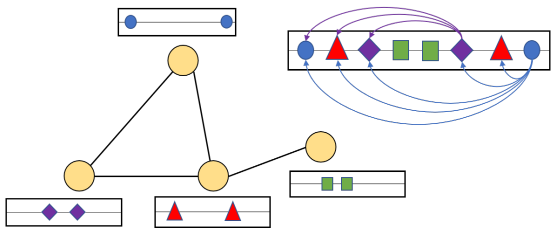
Figure 3: Illustration of event sequences on a graph. Sequences on vertices are aligned temporally to form a long sequence, and relational information among events are shown in arrows. Notice that only the structural information of the last event (the blue circle) and the third to the last event (the purple diamond) are shown. Events cannot attend to future.

highly smooth and ensures bias introduced by linear interpolations (Eq. 9) between consecutive events are small.

4 STRUCTURED TRANSFORMER HAWKES PROCESS

THP is quite general and can incorporate additional structural knowledge. We consider multiple point processes, where any two of them can be related. Such relationships are often described by a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ is the vertex set, and each vertex is associated with a point process. Also, $\mathcal{E}$ is the edge set, where each edge signifies relational information between the corresponding two vertices. Figure 3 illustrates event sequences on a graph.

The graph encodes relationships among vertices, and further indicates potential interactions. We propose to model all the point processes with a single THP, and the heterogeneity of the vertices' point processes is handled by a vertex embedding approach.

Suppose we have an event sequence $\mathcal{S} = \{(t_j, k_j, v_j)\}_{j=1}^{L}$, where t_j and k_j are time stamps and event types as before. Further, $v_j \in \{1, 2, \ldots, |\mathcal{V}|\}$ is an indicator to which vertex the event belongs. In addition to the event embedding and the temporal encoding (Eq. 2), we introduce a vertex embedding matrix $\mathbf{E} \in \mathbb{R}^{M \times |\mathcal{V}|}$, where the j-th column of $\mathbf{E}$ denotes the M-dimensional embedding for vertex j. Let $\mathbf{v}_j$ be the one-hot encoding of v_j, then embedding of $\mathcal{S}$ is specified by

$$\mathbf{X} = \left(\mathbf{UY} + \mathbf{EV} + \mathbf{Z}\right)^\top,$$

where $\mathbf{V} = [\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_L] \in \mathbb{R}^{|\mathcal{V}| \times L}$ is the concatenation of vertex embedding, and other terms are defined in Eq. 2.

The graph attention output is defined by

$$\mathbf{S} = \mathrm{Softmax}\left(\frac{\mathbf{QK}^\top}{\sqrt{M_K}} + \mathbf{A}\right)\mathbf{V}_{\text{value}},$$
$$\mathbf{A} = (\mathbf{EV})^\top \Omega (\mathbf{EV}), \tag{10}$$

where $\mathbf{Q}$, $\mathbf{K}$, and $\mathbf{V}_{\text{value}}$ are the same[2] as in Eq. 3. Matrix $\mathbf{A} \in \mathbb{R}^{L \times L}$ is a vertex similarity matrix, where each entry $\mathbf{A}_{ij}$ signifies the similarity between two vertices v_i and v_j, and $\Omega \in \mathbb{R}^{M \times M}$ is a metric to be learned. To extend the graph self-attention module to a multi-head setting, we use different metric matrices $\{\Omega_j\}_{j=1}^{H}$ for different heads.

[2]We use $\mathbf{V}_{\text{value}}$ to denote the value matrix instead of $\mathbf{V}$, which denotes the vertex embedding.

We remark that unlike RNN-based shallow models, in structured-THP, multiple multi-head self-attention modules can be stacked (Figure 2) to learn high level representations, a feature that enables learning of complicated similarities among vertices. Moreover, the vertex similarity matrix enables modeling of even more complicated structured data, such as sequences on dynamically evolving graphs.

With the incorporation of relational information, we need to modify the conditional intensity function accordingly. As an extension to Eq. 5, where each type of events has its own intensity, we define a different intensity function for each event type and each vertex. Specifically,

$$\lambda(t|\mathcal{H}_t) = \sum_{k=1}^{K}\sum_{v=1}^{|\mathcal{V}|} \lambda_{k,v}(t|\mathcal{H}_t), \quad t \in [t_j, t_{j+1}),$$
$$\lambda_{k,v}(t|\mathcal{H}_t) = f_{k,v}\Big(\alpha_{k,v}\frac{t - t_j}{t_j} + \mathbf{w}_{k,v}^\top \mathbf{h}(t) + b_{k,v}\Big).$$

Model parameters are learned by maximizing the log-likelihood (Eq. 7) across all sequences. Concretely, suppose we have N sequences $\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_N$, then parameters are obtained by solving

$$\max \sum_{i=1}^{N} \ell(\mathcal{S}_i) + \mu L_{\text{graph}}(\mathbf{V}, \Omega),$$

where μ is a hyper-parameter and

$$L_{\text{graph}}(\mathbf{V}, \Omega) = \sum_{k=1}^{|\mathcal{V}|}\sum_{j=1}^{k} -\log\left(1 + \exp(\mathbf{V}_j \Omega \mathbf{V}_k)\right)$$
$$+ \mathbf{1}\{(v_j, v_k) \in \mathcal{E}\}\big(\mathbf{V}_j \Omega \mathbf{V}_k\big).$$

Here $L_{\text{graph}}(\mathbf{V}, \Omega)$ is a regularization term that encourages $\mathbf{V}_j \Omega \mathbf{V}_k$ to be large when there exists an edge between v_j and v_k. Which means if two vertices are connected in graph $\mathcal{G}$, then the regularizer will promote attention between them, and vice versa.

Notice that in the simplest case, $\mathbf{A}$ in Eq. 10 can be some transformation of the adjacency matrix, i.e., $\mathbf{A}_{ij} = 1$ if $(v_i, v_j) \in \mathcal{E}$, and 0 otherwise. However, we believe that this constraint is too strict, i.e., some connected vertices may not be similar. Therefore, we treat the graph as a guide and introduce a regularization term that encourages $\mathbf{A}$ to be similar to the adjacency matrix, but not enforce it. In this way, our model is more flexible.

5 EXPERIMENTS

We compare THP against existing models: Recurrent Marked Temporal Point Process (RMTPP) [8], Neural Hawkes Process (NHP) [21], Time Series Event Sequence (TSES) [33], and Self-attentive Hawkes Processes (SAHP) [36][3]. We evaluate the models by per-event log-likelihood (in nats) and event prediction accuracy on held-out test sets.

5.1 Datasets

We adopt several datasets to evaluate the models. Table 1 summarizes statistics of the datasets.

Retweets [37]: The Retweets dataset contains sequences of tweets, where each sequence contains an origin tweet (i.e., some user initiates a tweet), and some follow-up tweets. We record the time and

[3]This is a concurrent work that also employs the Transformer architecture, and we only include results reported in their paper.

Table 1: Datasets statistics. From left to right columns: name of the dataset, number of event types, number of events in the dataset, and average length per sequence.

Dataset	K	# events	Avg. length
Retweets	3	2, 173, 533	109
MemeTrack	5000	123, 639	3
Financial	2	414, 800	2074
MIMIC-II	75	2, 419	4
StackOverflow	22	480, 413	72
911-Calls	3	290, 293	403
Earthquake	2	256, 932	500

the user tag of each tweet. Further, users are grouped into three categories based on the number of their followers: "small", "medium", and "large".

MemeTrack [18]: This dataset contains mentions of 42 thousand different memes spanning ten months. We collect data on over 1.5 million documents (blogs, web articles, etc.) from over 5000 websites. Each sequence in this dataset is the life-cycle of a particular meme, where each event (usage of meme) in the sequence is associated with a time stamp and a website id.

Financial Transactions [8]: This financial dataset contains transaction records of a stock in one day. We record the time (in milliseconds) and the action that was taken in each transaction. The dataset is a single long sequence with only two types of events: "buy" and "sell". The event sequence is further partitioned by time stamps.

Electrical Medical Records [16]: MIMIC-II medical dataset collects patients' visit to a hospital's ICU in a seven-year period. We treat the visits of each patient as a separate sequence, where each event in the sequence contains a time stamp and a diagnosis.

StackOverflow [18]: StackOverflow is a question-answering website. The website rewards users with badges to promote engagement in the community, and the same badge can be rewarded multiple times to the same user. We collect data in a two-year period, and we treat each user's reward history as a sequence. Each event in the sequence signifies receipt of a particular medal.

911-Calls[4]: The 911-Calls dataset contains emergency phone call records. Calling time, location of the caller, and nature of the emergency are logged for each record. We consider three types of emergencies: EMS, fire, and traffic. We treat location of callers (given by zipcodes) as vertices on a relational information graph. Zipcodes are ranked based on the number of recorded calls, and only the top 75 zipcodes are kept. An undirected edge exists between two vertices if their zipcodes are within 10 of each other.

Earthquake[5]: This dataset contains time and location of earthquakes in China in an eight-year period. We partition the records into two categories: "small" and "large". A relational information graph is built based on geographical locations of the earthquakes, i.e., each province is a vertex and earthquakes are sequences on the vertices. Two vertices are connected if their associated provinces are neighbors.

[4]The dataset is available on www.kaggle.com/mchirico/montcoalert.
[5]The dataset is provided by China Earthquake Data Center. (http://data.earthquake.cn)

Table 2: Log-likelihood comparison. Here RT is the Retweets dataset, MT is the MemeTrack dataset, FIN is the Financial Transactions dataset, and SO is the StackOverflow dataset.

Model	RT	MT	FIN	MIMIC-II	SO
RMTPP	-5.99	-6.04	-3.89	-1.35	-2.60
NHP	-5.60	-6.23	-3.60	-1.38	-2.55
SAHP	-4.56	—	—	-0.52	-1.86
THP	**-2.04**	**0.68**	**-1.11**	**0.820**	**0.042**

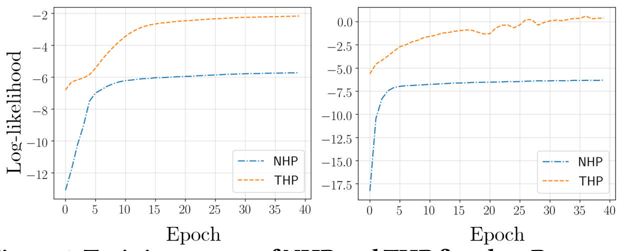

Figure 4: Training curves of NHP and THP fitted on Retweets (left figure) and MemeTrack (right figure).

5.2 Likelihood Comparison

We fit THP and NHP on Retweets and MemeTrack. From Figure 4, we can see that THP outperforms NHP during the entire training process by a large margin on both of the datasets. The reason is because of the complicated nature of social media data, and RNN-based models such as NHP are not powerful enough to model the dynamics.

In the Retweets dataset, we often observe time gaps between two consecutive retweets become larger, and this dynamic can be successfully modeled by temporal encoding. Also, unlike RNN-based models, our model is able to capture long-term dependencies that exist in long sequences. In the MemeTrack dataset, we have extremely short sequences, i.e., average sequence length is 3. Even though the data only exhibit short-term dependencies, we still need to model latent properties of memes such as topics and targeted users. We build deep THP models to capture these high-level features, and we remark that constructing deep NHP is not plausible because of the difficulty in training.

Table 2 summarizes results on other datasets. Note that TSES is likelihood-free. Our THP model fits the data well and outperforms all the baselines in all the experiments.

Figure 5 visualizes attention patterns of THP. We can see that each attention head employs a different pattern to capture dependencies. Moreover, while attention heads in the first layer tend to focus on individual events, the attention patterns in the last layer are more uniformly distributed. This is because features in deeper layers are already transformed by attention heads in shallow layers.

5.3 Event Prediction Comparison

For point processes, event prediction is just as important as data fitting. Eq. 6 enables us to predict future events. In practice, however, adding additional prediction layers on top of the THP model yields better performance. Specifically, given the hidden representation

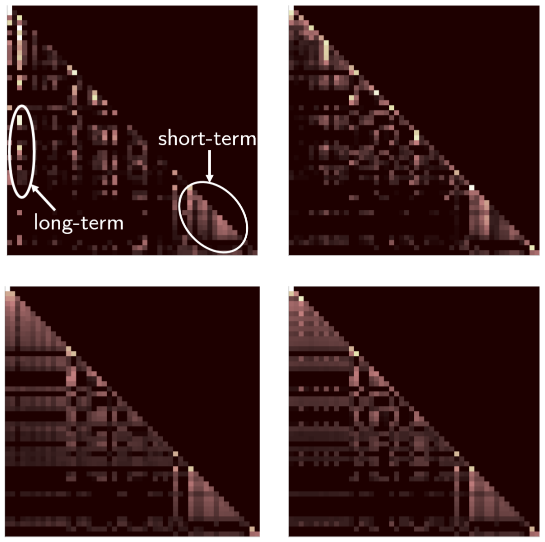

Figure 5: Visualization of attention patterns of different attention heads in different layers. Pixel (i, j) each figure is the attention weight of event t_j attending to event t_i. Attention heads in the upper two figures are from the first layer, while they are from the last layer in the lower two figures.

$\mathbf{h}(t_j)$ for event (t_j, k_j), the next event type and time predictions are as follows.

◇ The next event type prediction is

$$\widehat{\mathbf{p}}_{j+1} = \text{Softmax}\big(\mathbf{W}^{\text{type}}\mathbf{h}(t_j)\big),$$
$$\widehat{k}_{j+1} = \underset{k}{\text{argmax}}\ \widehat{\mathbf{p}}_{j+1}(k),$$

where $\mathbf{W}^{\text{type}} \in \mathbb{R}^{K\times M}$ is the parameter of the event type predictor, and $\widehat{\mathbf{p}}_j(k)$ is the k-th element of $\widehat{\mathbf{p}}_j \in \mathbb{R}^K$.

◇ The next event time prediction is

$$\widehat{t}_{j+1} = \mathbf{W}^{\text{time}}\mathbf{h}(t_j),$$

where $\mathbf{W}^{\text{time}} \in \mathbb{R}^{1\times M}$ is the predictor parameter.

To learn the predictor parameters, the loss function is equipped with a cross-entropy term for event type prediction and a mean square error term for event time prediction. Concretely, for an event sequence $\mathcal{S} = \{(t_j, k_j)\}_{j=1}^L$, let $\mathbf{k}_1, \mathbf{k}_2, \ldots, \mathbf{k}_L$ be the ground-truth one-hot encodings for the event types, we define

$$L_{\text{type}}(\mathcal{S}) = \textstyle\sum_{j=2}^L -\mathbf{k}_j^\top \log(\widehat{\mathbf{p}}_j),$$
$$L_{\text{time}}(\mathcal{S}) = \textstyle\sum_{j=2}^L (t_j - \widehat{t}_j)^2,$$

notice that we do not predict the first event. Then, given event sequences $\{\mathcal{S}_i\}_{i=1}^N$, we seek to solve

$$\min \sum_{i=1}^N -\ell(\mathcal{S}_i) + L_{\text{type}}(\mathcal{S}_i) + L_{\text{time}}(\mathcal{S}_i),$$

where $\ell(\mathcal{S}_i)$ is the log-likelihood (Eq. 7) of $\mathcal{S}_i$.

To evaluate model performance, we predict every held-out event (t_j, k_j) given its history $\mathcal{H}_j$, i.e., for a test sequence of length L, we make $L - 1$ predictions. We evaluate event type prediction by accuracy and event time prediction by Root Mean Square Error (RMSE). Table 3 and Table 4 summarize experiment results. We can see that THP outperforms the baselines in all these tasks. The datasets we adopted vary significantly in average sequence length, i.e., the average length in Financial Transactions is 2074 while it is only 4 in MIMIC-II. In all the three datasets, THP improves upon RNN-based models by a notable margin. The results demonstrate that THP is able to capture both short-term and long-term dependencies better than existing methods.

Table 3: Event type prediction accuracy comparison.

Model	Financial	MIMIC-II	StackOverflow
RMTPP	61.95	81.2	45.9
NHP	62.20	83.2	46.3
TSES	62.17	83.0	46.2
THP	**62.64**	**85.3**	**47.0**

Table 4: Event time prediction RMSE comparison.

Model	Financial	MIMIC-II	StackOverflow
RMTPP	1.56	6.12	9.78
NHP	1.56	6.13	9.83
TSES	1.50	4.70	8.00
SAHP	—	3.89	5.57
THP	**0.93**	**0.82**	**4.99**

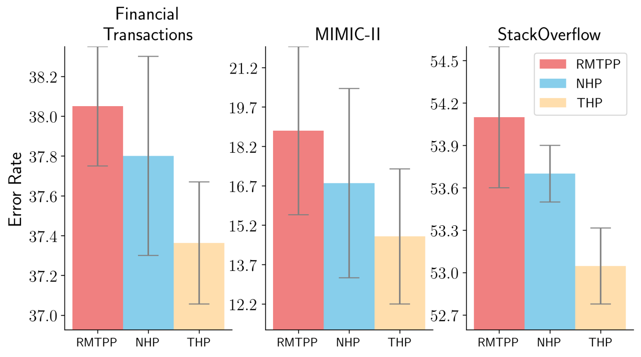

Figure 6: Prediction error rates of THP, NHP, and RMTPP. Based on a same train-dev-test splitting ratio, each dataset is sampled five times to produce different datasets. Error bars are generated according to these experiments.

Figure 6 illustrates run-to-run variance of THP, NHP, and RMTPP. The error bars are wide because of how the data are split. Held-out test sets are constructed by randomly sampling some events from the entire dataset. That is, at times "important" events are sampled out and that will yield unsatisfactory model performance. Our results are better than all the baselines in all the individual experiments.

5.4 THP vs. Structured-THP

Now we demonstrate by incorporating relational information, THP achieves improved performance.

Baseline models are constructed as following: for each vertex on a relational graph $\mathcal{G}$, there exists a point process that consists of time and type of events. These event sequences are learned separately by both THP and NHP, i.e., we do not allow information sharing among vertices in these models.

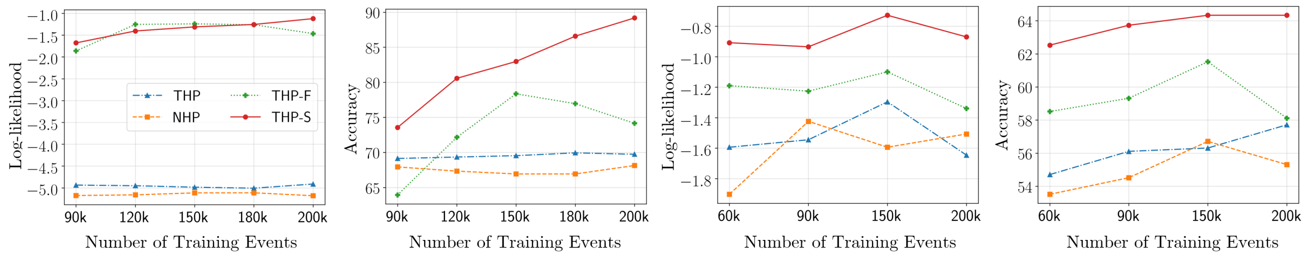

Figure 7: Log-likelihood and prediction accuracy of NHP, THP, THP with full attention (THP-F), and structured-THP (THP-S) fitted on the 911-Calls (left two figures) and the Earthquake (right two figures) datasets.

To integrate $\mathcal{G}$ into THP, we consider two approaches. The first approach is by allowing full attention, i.e., information from one vertex can be shared with all the other vertices. The second approach is by using the neighborhood graph, which is constructed based on spatial proximity. In this approach, a specific vertex can only share information with its neighbors. We fit a structured-THP to both of the cases.

Figure 7 summarizes experimental results. We can see that THP is comparable or better than NHP in both validation likelihood and event prediction, which further demonstrates that THP can model complicated dynamics better than RNN-based models. Notice that THP-F, the structured-THP with full attention, yields a much better likelihood than the baseline models, which means relational information sharing can help the models in capturing latent dynamics. However, unlike likelihood, THP-F does not show consistent improvements in event prediction. This is because when the number of training events is small, the model cannot build a sufficient information-sharing heuristic. Also, the performance drop when the number of training events is large is due to the inhomogeneity of data. This demonstrates that the full attention scheme results in undesirable dependencies on which the attention heads focus. THP-S successfully resolves this issue by eliminating such dependencies from the attention heads' span based on spatial closeness of vertices. In this way THP-S further improves upon THP-F, especially in event prediction tasks.

5.5 Ablation Study

We perform ablation study on Retweets and MemeTrack, and we evaluate models by validation log-likelihood. We inspect variants of THP by removing self-attention and temporal encoding mechanisms. Moreover, we test the effect of temporal encoding on NHP. Table 5 summarizes experimental results. As shown, both the self-attention module and the temporal encoding contribute to model performance.

We examine the models' sensitivity to the number of parameters on the Retweets dataset. As shown in Table 6, our model is not sensitive to its number of parameters. Without the recurrent structure, Transformer-based models often have large number of parameters, but our THP model can outperform RNN-based models with fewer parameters. In all the experiments, using a small model (about 100-200k parameters) will suffice. In comparison, NHP has about 1000k and TSES has about 2000k parameters to achieve the best performance, which are much larger than THP. We also include run-time comparison in Table 6. We conclude that THP is efficient in both model size and training speed.

Table 5: Log-likelihood of variants of NHP and THP fitted on Retweets and MemeTrack. TE stands for temporal encoding (Eq. 1), and PE stands for positional encoding [29].

Model	Retweets	MemeTrack
NHP	−5.60	−6.23
NHP + TE	−2.50	−1.64
Atten	−5.29	−5.09
Atten + PE	−5.25	−4.70
Atten + TE	**−2.03**	**0.68**

Table 6: Sensitivity to the number of parameters and run-time comparison. Speedup is the speed of THP against NHP.

# parameters	Log-likelihood		Speedup
	THP	NHP	
100k	−2.090	−6.019	×1.985
200k	−2.072	−5.595	×2.564
500k	−2.058	−5.590	×2.224
1000k	−2.060	−5.614	×1.778

6 CONCLUSION

In this paper we present Transformer Hawkes Process, a framework for analyzing event streams. Event sequence data are common in our daily life, and they exhibit sophisticated short-term and long-term dependencies. Our proposed model utilizes the self-attention mechanism to capture both of these dependencies, and meanwhile enjoys computational efficiency. Moreover, THP is quite general and can integrate structural knowledge into the model. This facilitates analyzing more complicated data, such as event sequences on graphs. Experiments on various real-world datasets demonstrate that THP achieves state-of-the-art performance in terms of both likelihood and event prediction accuracy.

ACKNOWLEDGMENTS

The work done by Haoming Jiang and Tuo Zhao is partially supported by NSF III 1717916. The work done by Hongyuan Zha is supported by Shenzhen Institute of Artificial Intelligence and Robotics for Society, and Shenzhen Research Institute of Big Data.

REFERENCES

[1] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. 2016. Layer normalization. *arXiv preprint arXiv:1607.06450* (2016).

[2] Emmanuel Bacry, Iacopo Mastromatteo, and Jean-François Muzy. 2015. Hawkes processes in finance. *Market Microstructure and Liquidity* 1, 01 (2015), 1550005.

[3] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473* (2014).

[4] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. 1994. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks* 5, 2 (1994), 157–166.

[5] Stephen P Borgatti, Ajay Mehra, Daniel J Brass, and Giuseppe Labianca. 2009. Network analysis in the social sciences. *science* 323, 5916 (2009), 892–895.

[6] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. 2014. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555* (2014).

[7] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805* (2018).

[8] 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.

[9] Mehrdad Farajtabar, Yichen Wang, Manuel Gomez-Rodriguez, Shuang Li, Hongyuan Zha, and Le Song. 2017. Coevolve: A joint point process model for information diffusion and network evolution. *The Journal of Machine Learning Research* 18, 1 (2017), 1305–1353.

[10] Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N Dauphin. 2017. Convolutional sequence to sequence learning. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, 1243–1252.

[11] Alan G Hawkes. 1971. Spectra of some self-exciting and mutually exciting point processes. *Biometrika* 58, 1 (1971), 83–90.

[12] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 770–778.

[13] Sepp Hochreiter, Yoshua Bengio, Paolo Frasconi, Jürgen Schmidhuber, et al. 2001. Gradient flow in recurrent nets: the difficulty of learning long-term dependencies.

[14] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation* 9, 8 (1997), 1735–1780.

[15] Valerie Isham and Mark Westcott. 1979. A self-correcting point process. *Stochastic Processes and Their Applications* 8, 3 (1979), 335–347.

[16] Alistair EW Johnson, Tom J Pollard, Lu Shen, H Lehman Li-wei, Mengling Feng, Mohammad Ghassemi, Benjamin Moody, Peter Szolovits, Leo Anthony Celi, and Roger G Mark. 2016. MIMIC-III, a freely accessible critical care database. *Scientific data* 3 (2016), 160035.

[17] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[18] Jure Leskovec and Andrej Krevl. 2014. SNAP Datasets: Stanford large network dataset collection.

[19] 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.

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

[21] 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.

[22] Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. 2016. Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499* (2016).

[23] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. 2013. On the difficulty of training recurrent neural networks. In *International conference on machine learning*. 1310–1318.

[24] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners. *OpenAI Blog* 1, 8 (2019).

[25] Christian Robert and George Casella. 2013. *Monte Carlo statistical methods*. Springer Science & Business Media.

[26] Sheldon M Ross, John J Kelly, Roger J Sullivan, William James Perry, Donald Mercer, Ruth M Davis, Thomas Dell Washburn, Earl V Sager, Joseph B Boyce, and Vincent L Bristow. 1996. *Stochastic processes*. Vol. 2. Wiley New York.

[27] Peter Shaw, Jakob Uszkoreit, and Ashish Vaswani. 2018. Self-attention with relative position representations. *arXiv preprint arXiv:1803.02155* (2018).

[28] Josef Stoer and Roland Bulirsch. 2013. *Introduction to numerical analysis*. Vol. 12. Springer Science & Business Media.

[29] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in neural information processing systems*. 5998–6008.

[30] D. Vere-Jones, Victoria University of Wellington. Institute of Statistics, and Operations Research. 1990. *Statistical Methods for the Description and Display of Earthquake Catalogues*. Victoria University of Wellington.

[31] Lu Wang, Wei Zhang, Xiaofeng He, and Hongyuan Zha. 2018. Supervised reinforcement learning with recurrent neural network for dynamic treatment recommendation. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM, 2447–2456.

[32] 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.

[33] Shuai Xiao, Junchi Yan, Xiaokang Yang, Hongyuan Zha, and Stephen M Chu. 2017. Modeling the intensity function of point process via recurrent neural networks. In *Thirty-First AAAI Conference on Artificial Intelligence*.

[34] Shuang-Hong Yang, Bo Long, Alex Smola, Narayanan Sadagopan, Zhaohui Zheng, and Hongyuan Zha. 2011. Like like alike: joint friendship and interest propagation in social networks. In *Proceedings of the 20th international conference on World wide web*. 537–546.

[35] Wenpeng Yin, Katharina Kann, Mo Yu, and Hinrich Schütze. 2017. Comparative study of CNN and RNN for natural language processing. *arXiv preprint arXiv:1702.01923* (2017).

[36] Qiang Zhang, Aldo Lipani, Omer Kirnap, and Emine Yilmaz. 2019. Self-attentive Hawkes processes. *arXiv preprint arXiv:1907.07561* (2019).

[37] Qingyuan Zhao, Murat A Erdogdu, Hera Y He, Anand Rajaraman, and Jure Leskovec. 2015. 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*. ACM, 1513–1522.

[38] Ke Zhou, Hongyuan Zha, and Le Song. 2013. Learning social infectivity in sparse low-rank networks using multi-dimensional hawkes processes. In *Artificial Intelligence and Statistics*. 641–649.