# LEARNING MIXTURE OF NEURAL TEMPORAL POINT PROCESSES FOR EVENT SEQUENCE CLUSTERING

**Anonymous authors**
Paper under double-blind review

## ABSTRACT

Event sequence clustering applies to many scenarios e.g. e-Commerce and electronic health. Traditional clustering models fail to characterize complex real-world processes due to the strong parametric assumption. While Neural Temporal Point Processes (NTPPs) mainly focus on modeling similar sequences instead of clustering. To fill the gap, we propose Mixture of Neural Temporal Point Processes (NTPP-MIX), a general framework that can utilize many existing NTPPs for event sequence clustering. In NTPP-MIX, the prior distribution of coefficients for cluster assignment is modeled by a Dirichlet distribution. When the assignment is given, the conditional probability of a sequence is modeled by the mixture of series of NTPPs. We combine variational EM algorithm and Stochastic Gradient Descent (SGD) to efficiently train the framework. Moreover, to further improve its capability, we propose a fully data-driven NTPP based on the attention mechanism named Fully Attentive Temporal Point Process (FATPP). Experiments on both synthetic and real-world datasets show the effectiveness of NTPP-MIX against state-of-the-arts, especially when using FATPP as a basic NTPP module.

## 1 INTRODUCTION

Many real-world processes can be characterized by time-stamped multi-typed event sequences, e.g. e-Commerce (Xu et al., 2014) and electronic health records (Enguehard et al., 2020). Given a set of event sequences, a natural task is to cluster them into groups such that sequences generated by similar underlying dynamics are grouped. Event sequence clustering is important to many downstream applications. For instance, clustering user's behaviors in e-Commerce helps to categorize user habits to promote recommendation system.

Despite the importance, research on event sequence clustering is very limited. Xu & Zha (2017) propose a Dirichlet Mixture model of Hawkes Processes (DMHP) where event sequences are characterized via Hawkes processes (Hawkes, 1971) with different parameters. However, as a traditional Temporal Point Process (TPP), it is hard for Hawkes process to model complex real-world dynamics due to its strong parametric assumption. Wu et al. (2020) take a reinforcement learning view of event sequence clustering. But it only works for univariate event sequences.

With the development of deep learning, many Neural Temporal Point Processes (NTPPs) have been developed. Recurrent Neural Networks (RNN) (Du et al., 2016; Mei & Eisner, 2016; Omi et al., 2019), Neural Ordinary Differential Equations (Jia & Benson, 2019; Chen et al., 2021) and Transformers (Zhang et al., 2020; Zuo et al., 2020) are used for event sequence modeling. Many Maximum Likelihood Estimation free (MLE-free) NTPPs are also developed (Xiao et al., 2017; Upadhyay et al., 2018; Yan et al., 2018; Li et al., 2018). Although these NTPPs are more flexible than Hawkes process, they can not perform clustering because they assume all observed sequences are generated by similar dynamics.

We propose mixture of neural temporal point processes (NTPP-MIX), a general framework which can incorporate various MLE-based NTPPs for event sequence clustering. Specifically, the prior of the mixing coefficients is modeled by a Dirichlet distribution. Then the coefficients are used to model the prior of cluster assignment. For each sequence, its conditional probability on the given assignment is modeled via series of NTPPs. We also propose a new algorithm based on variational EM and Stochastic Gradient Descent (SGD) to train the framework. Moreover, we develop our Fully

Attentive Temporal Point Process (FATPP) which is a fully data-driven NTPP without parametric assumption to further improve NTPP-MIX. **The contributions of our work are:**

**1)** We develop a general framework NTPP-MIX and a corresponding training algorithm for event sequence clustering. NTPP-MIX is the first neural-network-based general framework for multi-typed event sequence clustering and can utilize various previous MLE-based NTPPs as its modules.

**2)** We also propose FATPP based on attention mechanism to further improve NTPP-MIX. Different from the peer methods (Zuo et al., 2020; Zhang et al., 2020; Gu, 2021) where attention is only used for encoding the event sequence, the intensity function in FATPP is also formulated by attention without parametric assumption. The hope is that this flexible model can better fit with various dynamics to benefit the clustering of mixed sequences in the real world.

**3)** Experiments on synthetic and real-world benchmarks show the effectiveness of our NTPP-MIX framework (especially with FATPP for modeling each cluster) against state-of-the-art methods.

## 2 Preliminaries and Related Works

### 2.1 Traditional temporal point processes

Temporal Point Process (TPP) (Daley & David, 2007) is a useful tool to model a set of similar event sequences. We denote an event sequence as $S = \{(t_i, v_i)\}_{i=1}^{L}$ i, where $t_i \in [0, T)$ is the timestamp and $v_i \in \{1, 2, \dots, C\}$ is the event type. A TPP first define the conditional intensity function:

$$\lambda(t, u|\mathcal{H}_t) = \lim_{dt \to 0} \frac{\mathbb{P}\left(N_u(t + dt) - N_u(t) = 1|\mathcal{H}_t\right)}{dt} \tag{1}$$

where $N_u(t)$ denotes the counting process which counts the number of events of type $u$ until time $t$, $\mathcal{H}_t = \{(t_i, v_i) : t_i < t\}$ represents the history up to $t$. The log likelihood of $S$ is defined as:

$$\log p(S) = \sum_{i=1}^{L} \log \lambda(t_i, v_i|\mathcal{H}_{t_i}) - \sum_{u=1}^{C} \int_0^{t_L} \lambda(t, u|\mathcal{H}_t)dt \tag{2}$$

Most traditional TPPs design a parametric form for intensity, which is often trained via MLE.

### 2.2 Neural temporal point processes

Neural Temporal Point Processes (NTPPs) have been recently proposed to capture complex dynamics. Du et al. (2016) use RNNs as the backbone and propose the first NTPP. Mei & Eisner (2016) further improve NTPP by constructing continuous-time RNNs where hidden state exponentially changes with time. Jia & Benson (2019); Chen et al. (2018) use the continuous-depth Neural Ordinary Differential Equations (Chen et al., 2018) to achieve more flexible continuous-time state evolution NTPPs (Shchur et al., 2021). Transformer or Attention mechanism is used in (Zhang et al., 2020; Zuo et al., 2020) to capture long-term dependency and improve training speed.

The above NTPPs use networks to model the conditional intensity and then optimize networks via MLE. There are also intensity-free (Shchur et al., 2020) and MLE-free (Xiao et al., 2017; Yan et al., 2018; Li et al., 2018) NTPPs.

Although these NTPPs are more flexible and generalized than traditional TPPs, they can not directly perform clustering as they assume observed sequences are generated by similar dynamics.

### 2.3 Event sequence clustering methods

Although event sequence modeling is well-studied, there are few studies about event sequence clustering. Wu et al. (2020) propose a reinforcement learning model but it only works for univariate event sequences (e.g. sequence without type mark $v_i$) and can not select the number of clusters automatically. To our best knowledge, the most related work to ours is DMHP (Xu & Zha, 2017) which mixes several Hawkes processes for event sequence clustering. Different clusters are defined as Hawkes processes with different parameters. However, real-world data may be complex and can not be well modeled by Hawkes processes. For instance, occur of an event can inhibit other events instead of the triggering assumption of Hawkes processes.

## 3 NTPP-MIX: Mixture of NTPPs for Event Clustering

In this section, we propose mixture of neural temporal point processes (NTPP-MIX) framework which combines neural temporal point process and variational inference for event sequence clustering. The framework is described in Sec. 3.1. An efficient training algorithm is proposed in Sec. 3.2.

### 3.1 Model Framework

Given a set of event sequences $\mathbf{S} = \{S_n\}_{n=1}^N$, where $S_n = \{(t_i, v_i)\}_{i=1}^{L_n}$ is a sequence defined in Sec. 2.1. Our goal is to divide these $N$ sequences into $K$ groups such that the sequences generated by similar dynamics are grouped. We assume for each $S_n$, there is a corresponding hidden vector $\mathbf{z}_n \in \{0,1\}^K, \sum_{k=1}^K z_{nk} = 1$ denoting the group it belongs to, i.e. $z_{nk} = 1$ *iff* $S_n$ belongs to group $k$. Denote the hidden variables as $\mathbf{Z} = \{\mathbf{z}_1, \dots, \mathbf{z}_N\}$ and our goal is to give a probabilistic estimation of $\mathbf{Z}$. We combine NTPPs and variational inference to enable a probabilistic framework for this clustering problem. Our NTPP-MIX is general and can utilize various NTPPs as its basic component. Fig. 1 shows an overview of NTPP-MIX.

We first assume the prior of mixing coefficients $\boldsymbol{\pi}$ follows a Dirichlet distribution:

$$p(\boldsymbol{\pi}) = \mathrm{Dir}(\boldsymbol{\pi}|\boldsymbol{\alpha}_0) \tag{3}$$

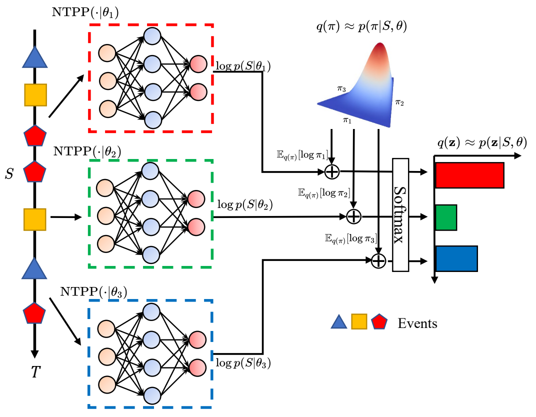
where $\boldsymbol{\alpha}_0 = [\alpha_0, \dots, \alpha_0] \in \mathbb{R}^K$. Setting a prior for $\boldsymbol{\pi}$ results in a variational model that helps to select $K$ (Bishop & Nasrabadi, 2007).

Given mixing coefficient $\boldsymbol{\pi}$, the conditional distribution of $\mathbf{Z}$ is formed as: $p(\mathbf{Z}|\boldsymbol{\pi}) = \prod_{n=1}^N \prod_{k=1}^K \pi_k^{z_{nk}}$ where $\pi_k$ is the $k$-th dimension of $\boldsymbol{\pi}$. This equals to $p(z_{nk} = 1|\boldsymbol{\pi}) = \pi_k$.

Given $\mathbf{Z}$, we use $K$ NTPPs to model the conditional probability of $\mathbf{S}$:

$$\begin{aligned} p(\mathbf{S}|\mathbf{Z}, \theta) &= \prod_{n=1}^N \prod_{k=1}^K p(S_n|\theta_k)^{z_{nk}} \\ p(S_n|\theta_k) &= \mathrm{NTPP}(S_n|\theta_k) \end{aligned} \tag{4}$$

where $\theta = \{\theta_k\}_{k=1}^K$ and each $\theta_k$ denotes parameters of the $k$-th NTPP model. For an event sequence $S_n$, we input it into the $k$-th NTPP, and get the conditional probability $p(S_n|\theta_k)$.



NTPP$(\cdot|\theta_1)$
NTPP$(\cdot|\theta_2)$
NTPP$(\cdot|\theta_3)$
$S$
$T$
$\log p(S|\theta_1)$
$\log p(S|\theta_2)$
$\log p(S|\theta_3)$
$q(\pi) \approx p(\pi|S,\theta)$
$\pi_1$
$\pi_2$
$\pi_3$
$\mathbb{E}_{q(\pi)}[\log \pi_1]$
$\mathbb{E}_{q(\pi)}[\log \pi_2]$
$\mathbb{E}_{q(\pi)}[\log \pi_3]$
Softmax
$q(\mathbf{z}) \approx p(\mathbf{z}|S,\theta)$
Events

Figure 1: Our NTPP-MIX framework for $K = 3$. After training, an event sequence is input into $K$ NTPPs to get conditional probabilities. Adding conditional probabilities with corresponding expectation of mixing coefficients, we can get the approximated posterior of assignment. Index $n$ is omitted. Details of the derivation is in Sec. 3.2.

Note that NTPP here can be any MLE-based neural temporal point process such as RMTPP (Du et al., 2016), NHP (Mei & Eisner, 2016), SAHP (Zhang et al., 2020), THP (Zuo et al., 2020), etc.

Summarizing above equations, we can get the joint distribution $p(\mathbf{S}, \mathbf{Z}, \boldsymbol{\pi}|\theta)$. Our goal is to maximize marginal distribution $p(\mathbf{S}|\theta)$ with respect to $\theta$ and get the posterior distribution $p(\mathbf{Z}|\mathbf{S}, \theta_{opt})$[1]:

$$\begin{aligned} \theta_{opt} &= \arg\max_\theta p(\mathbf{S}|\theta) = \arg\max_\theta \iint p(\mathbf{S}, \mathbf{Z}, \boldsymbol{\pi}|\theta) d\mathbf{Z} d\boldsymbol{\pi} \\ p(\mathbf{S}, \mathbf{Z}, \boldsymbol{\pi}|\theta) &= p(\boldsymbol{\pi}) p(\mathbf{Z}|\boldsymbol{\pi}) p(\mathbf{S}|\mathbf{Z}, \theta) = \mathrm{Dir}(\boldsymbol{\pi}|\boldsymbol{\alpha}_0) \prod_{n=1}^N \prod_{k=1}^K [\pi_k \mathrm{NTPP}(S_n|\theta_k)]^{z_{nk}} \end{aligned} \tag{5}$$

### 3.2 Training Algorithm

It is hard to optimize marginal probability $p(\mathbf{S}|\theta)$ and get the posterior $p(\mathbf{Z}|\mathbf{S}, \theta_{opt})$ directly as the model contains a probability term $p(S_n|\theta_k)$ modeled by neural-network-based models. To address this problem, we use a variational inference framework for training. We introduce a variational distribution $q(\mathbf{Z}, \boldsymbol{\pi})$ to approximate $p(\mathbf{Z}, \boldsymbol{\pi}|\mathbf{S}, \theta)$. Using $q(\mathbf{Z}, \boldsymbol{\pi})$, we can get:

$$\mathcal{L}(q, \theta) = \log p(\mathbf{S}|\theta) - \mathrm{KL}(q\|p) \tag{6}$$

[1] $\mathbf{Z}$ is discrete and we use integration for simplification

where we define:

$$\begin{aligned}\mathcal{L}(q,\theta) &= \iint q(\mathbf{Z},\boldsymbol{\pi}) \log\left[\frac{p(\mathbf{S},\mathbf{Z},\boldsymbol{\pi}|\theta)}{q(\mathbf{Z},\boldsymbol{\pi})}\right] d\mathbf{Z}d\boldsymbol{\pi} \\ \mathrm{KL}(q\|p) &= \iint q(\mathbf{Z},\boldsymbol{\pi}) \log\left[\frac{q(\mathbf{Z},\boldsymbol{\pi})}{p(\mathbf{Z},\boldsymbol{\pi}|\mathbf{S},\theta)}\right] d\mathbf{Z}d\boldsymbol{\pi}\end{aligned} \tag{7}$$

Therefore, maximizing $\mathcal{L}(q,\theta)$ with respect to $q$ and $\theta$ is equivalent to maximizing $\log p(\mathbf{S}|\theta)$ and minimizing $\mathrm{KL}(q\|p)$. Maximizing $\log p(\mathbf{S}|\theta)$ makes our model characterize observed sequences well. Minimizing $\mathrm{KL}(q\|p)$ gives an approximation of posterior. Then our goal changes to:

$$\max_{\theta,q} \mathcal{L}(q,\theta) \tag{8}$$

Using mean field approximation (Opper & Saad, 2001), we assume $q(\mathbf{Z},\boldsymbol{\pi}) = q(\mathbf{Z})q(\boldsymbol{\pi})$ and maximize $\mathcal{L}(q,\theta)$ with respect to $q(\mathbf{Z}), q(\boldsymbol{\pi}), \theta$ iteratively according to the following process. The detailed derivation is presented in Appendix A.

**Update $q(\mathbf{Z})$ (E-step)** Fixing $q(\boldsymbol{\pi})$ and $\theta$, the log of the optimized $q(\mathbf{Z})$ is give by:

$$\begin{aligned}\log q^*(\mathbf{Z}) =& \mathbb{E}_{q(\boldsymbol{\pi})}[\log p(\mathbf{S},\mathbf{Z},\boldsymbol{\pi}|\theta)] + \mathrm{Const} = \mathbb{E}_{q(\boldsymbol{\pi})}[\log p(\mathbf{Z}|\boldsymbol{\pi})] + \log p(\mathbf{S}|\mathbf{Z},\theta) + \mathrm{Const} \\ =& \sum_{n=1}^{N}\sum_{k=1}^{K} z_{nk} \underbrace{\left\{\mathbb{E}_{q(\boldsymbol{\pi})}[\log \pi_k] + \log p(S_n|\theta_k)\right\}}_{\log \rho_{nk}} + \mathrm{Const}\end{aligned} \tag{9}$$

Normalizing the above formulation, we obtain:

$$q^*(\mathbf{Z}) = \prod_{n=1}^{N}\prod_{k=1}^{K} r_{nk}^{z_{nk}}, \quad r_{nk} = \frac{\rho_{nk}}{\sum_{j=1}^{K}\rho_{nj}} \tag{10}$$

As $q(\mathbf{Z})$ is an approximation of $p(\mathbf{Z}|\mathbf{S},\theta)$, when the model is well-trained, $p(z_{nk}=1|S_n,\theta_{opt}) \approx r_{nk}$, i.e. the posterior probability that $S_n$ in group $k$ is $r_{nk}$.

**Update $q(\boldsymbol{\pi})$ (M-step)** Fixing $q(\mathbf{Z})$ and $\theta$, the log of the optimized $q(\boldsymbol{\pi})$ is give by:

$$\begin{aligned}\log q^*(\boldsymbol{\pi}) =& \mathbb{E}_{q(\mathbf{Z})}[\log p(\mathbf{S},\mathbf{Z},\boldsymbol{\pi}|\theta)] + \mathrm{Const} = \log p(\boldsymbol{\pi}) + \mathbb{E}_{q(\mathbf{Z})}[\log p(\mathbf{Z}|\pi)] + \mathrm{Const} \\ =& \sum_{k=1}^{K}\left(\alpha_0 + \sum_{n=1}^{N} r_{nk} - 1\right)\log \pi_k + \mathrm{Const}\end{aligned} \tag{11}$$

Normalizing the above formulation, we recognize $q^*(\boldsymbol{\pi})$ is a Dirichlet distribution:

$$q^*(\boldsymbol{\pi}) = \mathrm{Dir}(\boldsymbol{\pi}|\boldsymbol{\alpha}), \quad \alpha_k = \alpha_0 + N_k, \quad N_k = \sum_{n=1}^{N} r_{nk} \tag{12}$$

**Update $\theta$ (M-step)** Fixing $q(\mathbf{Z})$ and $q(\boldsymbol{\pi})$, optimize $\mathcal{L}(q,\theta)$ w.r.t. $\theta$. Since $\theta$ are parameters for NTPPs, it is hard to get the optimal form like $q(\mathbf{Z})$ and $q(\boldsymbol{\pi})$, we use SGD to optimize the following objective function:

$$\mathcal{L}(\theta) = \mathbb{E}_{q(\mathbf{Z})}[\log p(\mathbf{S}|\mathbf{Z},\theta)] + \mathrm{Const} = \sum_{n=1}^{N}\sum_{k=1}^{K} r_{nk}\log p(S_n|\theta_k) + \mathrm{Const} \tag{13}$$

where $\mathcal{L}(\theta)$ denotes the term in $\mathcal{L}(q,\theta)$ that is only related to $\theta$.

**Update the cluster number $K$** When $K$ is unknown, we initialize $K$ as a large number and remove redundant clusters during training. In our variational framework, $q^*(\boldsymbol{\pi})$ is an approximation of $p(\boldsymbol{\pi}|\mathbf{S},\theta)$, we can update $K$ based on $q^*(\boldsymbol{\pi})$. In each iteration, we first run E-step and M-step for optimization. After optimization, we can get $q^*(\boldsymbol{\pi})$ and its parameter $\boldsymbol{\alpha}$. A small $\alpha_k$ indicates the effective number of sequences corresponding to cluster $k$ is small, so we remove it. The removement also helps to speed up the training process as the number of NTPPs to fit is decreasing.

Detailed training is shown in Algorithm 1. We use mini-batch training for efficient optimization and use multiple SGD steps in one M-step.

**Algorithm 1:** Variational EM training algorithm for NTPP-MIX

```
Input: Sequences $\mathbf{S} = \{S_n\}_{n=1}^N$. Initial number of clusters $K$ (to be automatically updated).
       Mini-batch size $B$. Training epochs $T$. Number of SGD steps to perform in each
       M-step $M$. $q(\boldsymbol{\pi})$ update rate $\eta$. Minimum cluster size $\epsilon$. Weights for prior $p(\boldsymbol{\pi})$: $\alpha_0$.
Output: Cluster assignment $q^*(\mathbf{Z}) \approx p(\mathbf{Z}|\mathbf{S}, \theta_{opt})$.
1  Initialize $\{\theta_k\}_{k=1}^K$ for $K$ NTPPs. Initialize $\{\alpha_k = \frac{\alpha_0+N}{K}\}_{k=1}^K$ for $q(\boldsymbol{\pi})$;
2  Pre-train each NTPP on $\mathbf{S}$ by MLE to prevent bad initialization;
3  for $epoch = 1, \cdots, T$ do
4      for $iter = 1, \cdots, \lceil \frac{N}{B} \rceil$ do
5          Sample a batch of sequences $\mathbf{S}'$ from $\mathbf{S}$;
           // E-step:  update $q(\mathbf{Z}')$
6          $\mathbb{E}_{q(\boldsymbol{\pi})}[\log \pi_k] = \text{Digamma}(\alpha_k) - \text{Digamma}(\sum_{k=1}^K \alpha_k) \quad k \in \{1, \dots, K\}$
7          $\log p(S_n|\theta_k) = \log[\text{NTPP}(S_n|\theta_k)] \quad S_n \in \mathbf{S}', k \in \{1, \dots, K\}$
8          $r_{nk} = \frac{\exp\left(\mathbb{E}_{q(\boldsymbol{\pi})}[\log \pi_k] + \log p(S_n|\theta_k)\right)}{\sum_{k=1}^K \exp\left(\mathbb{E}_{q(\boldsymbol{\pi})}[\log \pi_k] + \log p(S_n|\theta_k)\right)} \quad S_n \in \mathbf{S}', k \in \{1, \dots, K\}$
           // M-step:  update $q(\boldsymbol{\pi})$
9          $\alpha_k = (1-\eta)\alpha_k + \eta(\alpha_0 + \frac{N}{|\mathbf{S}'|} \sum_{S_n \in \mathbf{S}'} r_{nk}) \quad k \in \{1, \dots, K\}$
           // M-step:  update $\theta$
10         for $step = 1, \cdots, M$ do
11             $\widetilde{\mathcal{L}}(\theta) = \sum_{S_n \in \mathbf{S}'} \sum_{k=1}^K r_{nk} \log p(S_n|\theta_k)$;
12             Update $\theta$ using SGD with $-\widetilde{\mathcal{L}}(\theta)$ as loss function;
       // Check and remove redundant clusters
13     $\forall k$, remove $\theta_k$ and distribute $\alpha_k$ to other clusters equally if $\alpha_k < \epsilon$;
14 Compute $r_{nk}, n \in \{1, \dots, N\}, k \in \{1, \dots, K\}$ using the trained model (line 6 to 8);
15 Return $q^*(z_{nk} = 1) = r_{nk}$;
```

## 4 FATPP: Fully Attentive Temporal Point Process

In the clustering scenario, we hope basic NTPP modules are flexible enough to model diverse processes. However, most NTPPs still make parametric assumptions about the intensity formulation, which limits the flexibility and affects the performance of NTPP-MIX. To fill the gap, we propose Fully Attentive Temporal Point Process (FATPP), a universal data-driven NTPP for event sequence modeling in this section. The overview of FATPP is shown in Fig. 2. In Sec. 4.1, we encode the sequence into hidden vectors. In Sec. 4.2, we use a general method to formulate the intensity.

### 4.1 History Events Encoding

In this section, we encode event sequence $S_n = \{(t_i, v_i)\}_{i=1}^{L_n}$ into hidden vectors to extract features (Fig. 2 left). We first embedding the $(t_i, v_i)$, then use self-attention for encoding.

**Event embedding.** We add the type and timestamp embedding to obtain the embedding of $(t_i, v_i)$:

$$\mathbf{x}_i = \text{TYPE}(v_i) + \text{TIME}(i, t_i)$$
$$\text{TYPE}(v) = \mathbf{W}_{type}\mathbf{e}^v, \quad \text{TIME}(i,t)[k] = \begin{cases} \sin(\phi_k i + \omega_k t) & k\%4 = 0 \\ \cos(\phi_k i + \omega_k t) & k\%4 = 1 \\ \text{Tanh}(\beta_k t) & k\%4 = 2, 3 \end{cases} \tag{14}$$

where $\mathbf{W}_{type} \in \mathbb{R}^{d \times C}$ is a learnable embedding matrix and $\mathbf{e}^v$ is a one-hot matrix. $\phi_k$ is a fixed parameter and $\omega_k, \beta$ are learnable parameters. The timestamp embedding is similar to Zhang et al. (2020) except we add Tanh term to capture the non-periodicity pattern.

**Self-Attention encoding.** After embedding the sequence $S_n = \{(t_i, v_i)\}_{i=1}^{L_n}$ into $\mathbf{X} = [\mathbf{x}_1; \mathbf{x}_2; \dots; \mathbf{x}_L] \in \mathbb{R}^{L \times d}$, we use Self-Attention to encode $\mathbf{X}$ for feature extraction:

$$\mathbf{Y} = \text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) \quad \text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{Softmax}(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d}})\mathbf{V}$$
$$\mathbf{Q} = \mathbf{X}\mathbf{W}^Q, \mathbf{K} = \mathbf{X}\mathbf{W}^K, \mathbf{V} = \mathbf{X}\mathbf{W}^V \tag{15}$$

where $\mathbf{W}^Q, \mathbf{W}^K, \mathbf{W}^V \in \mathbb{R}^{d\times d}$ are learnable parameters, $\mathbf{Y} = [\mathbf{y}_1; \mathbf{y}_2; \ldots; \mathbf{y}_L] \in \mathbb{R}^{L\times d}$. In practice, we use multi-head attention. Masking is used to prevent getting future information.

### 4.2 Attention-based Universal Intensity Formulation

Having hidden vectors $\{\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_L\}$, we now use these hidden vectors to formulate the intensity function $\lambda(t, u|\mathcal{H}_t)$, where $t \in (t_j, t_{j+1}]$ is the timestamp to compute intensity, $u$ is the event type.

Previous attention-based works design different forms for intensity: Zuo et al. (2020) assume that the intensity changes linearly with time (ignoring softplus), Zhang et al. (2020) and Gu (2021) assume it changes exponentially. These assumptions limit the generalization of attention as the intensity is limited to fixed forms. One obvious limitation of the above three formulations is that the monotonicity of the intensity can not change, i.e. the intensity between two events is either increasing or decreasing. This can hardly hold in real-world cases especially for scattered events.

We use an attention based method to formulate the intensity. We make no assumption in our method and how intensity changes is fully learned from data. The main idea is using $(t, u)$ to "query the past" as sketched in the right part of Fig. 2. Formally, we first embed $(t, u)$(where we want to compute the intensity) into a hidden vector irrelevant to the past using Eq. 14:

$$\mathbf{h}(t, u) = \text{TYPE}(u) + \text{TIME}(j + 1, t) \quad (16)$$

Then $\mathbf{h}(t, u)$ is used as the query in attention mechanism and $\mathbf{Y}_{[:j]} = [\mathbf{y}_1; \mathbf{y}_2; \ldots; \mathbf{y}_j]$ is used as keys and values to generate a hidden vector relevant to the past:

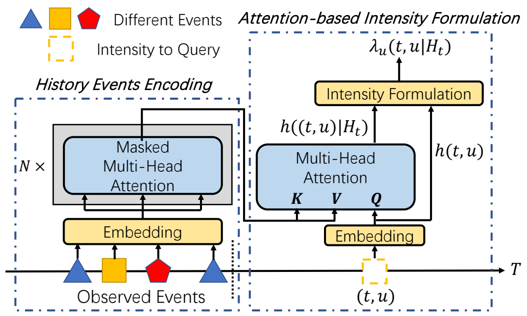


Figure 2: Overview of FATPP: we use attention to encode history (left). Then intensity at any time can be flexibly modeled by query (right).

$$\begin{gathered}\mathbf{h}((t, u)|\mathcal{H}_t) = \text{Attention}(\mathbf{q}, \mathbf{K}_h, \mathbf{V}_h) \\ \mathbf{q} = \mathbf{W}_h^Q \mathbf{h}(t, u), \quad \mathbf{K}_h = \mathbf{Y}_{[:j]}\mathbf{W}_h^K, \quad \mathbf{V} = \mathbf{Y}_{[:j]}\mathbf{W}_h^V\end{gathered} \tag{17}$$

In practice, we use multi-head attention. $\mathbf{h}(t, u)$ and $\mathbf{h}((t, u)|\mathcal{H}_t)$ are used to formulate intensity:

$$\lambda(t, u|\mathcal{H}_t) = f\left(\mathbf{w}_{base}^\top \mathbf{h}(t, u) + \mathbf{w}_{eff}^\top \mathbf{h}((t, u)|\mathcal{H}_t)\right) \tag{18}$$

Here $f$ is the softplus function, $\mathbf{w}_{base}$ and $\mathbf{w}_{eff}$ are parameters to learn. The first term is only relevant to current time and type and the second term reflects the effect of past events. There is no explicit time $t$ as it is embedded into the two hidden vectors. This means that we make no assumption about how intensity changes over time. Our FATPP is more flexible than semi-parametric NTPPs and can model more diverse processes. Therefore, it is more suitable for our NTPP-MIX framework. The efficiency analysis for FATPP compared with other Attention-based NTPP is in Appendix C.

From the perspective of intensity formulation, our FATPP is similar to Mei & Eisner (2016); Jia & Benson (2019) and can be viewed as a continuous-time state evolution NTPP (Shchur et al., 2021): at any time and type $(t, u)$, there are corresponding vectors representing states on continuous time domain. The intensity formulation process is also similar to the Decoder in Transformer (Vaswani et al., 2017), but our model "decodes" intensity at any timestamp and the length of output is unfixed.

## 5 Experiments

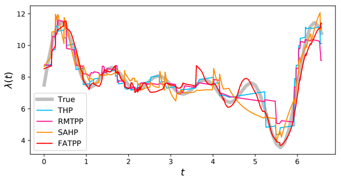
We conduct experiments on both synthetic and real-world datasets. Sec. 5.1 compares the flexibility of FATPP with other NTPPs. In Sec. 5.2 we evaluate the clustering ability of NTPP-MIX.

### 5.1 Experiments on single NTPPs for event sequence modeling

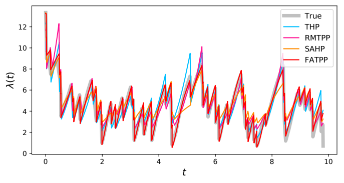
In this section, we compare FATPP with other NTPPs on modeling observed sequences to verify its flexibility for modeling diverse processes. We generate five synthetic datasets using different processes: **1) Homo** Homogeneous Possion process. **2) In-homo** In-homogeneous Possion process. **3) Inhibit** Process in which events of different types inhibit each other. **4) Excite** Process in which events of different types excite each other (Hawkes). **5) In&Ex** Process in which relation is either inhibition or excitation. Each dataset contains 4,000 event sequences, and each sequence contains $L_n = 50$ events with number of event types $C = 5$.

Table 1: Average log likelihood estimation of single NTPPs on synthetic and real-world datasets. Note that time intervals in eCommerce vary from seconds to hours, which makes THP crash.

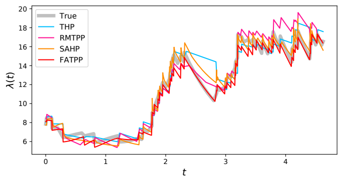
| Dataset | Homo | In-homo | Inhibit | Excite | In&Ex | Stock | eCommerce | Neuron |
|---|---|---|---|---|---|---|---|---|
| RMTPP | -23.81 | -23.95 | -42.56 | -40.46 | 26.07 | -98.44 | -187.10 | 49.99 |
| THP | -23.80 | -23.89 | -41.35 | -39.90 | **26.91** | -92.06 | N/A | 48.26 |
| SAHP | -23.70 | -23.23 | -43.63 | -39.71 | 25.87 | -93.64 | **-121.86** | 66.34 |
| LogNormMix | -23.50 | -23.52 | -42.01 | -39.74 | 26.25 | -77.47 | -136.70 | **68.23** |
| AMDN | **-23.38** | -23.88 | -42.09 | **-39.60** | 25.95 | -77.19 | -148.00 | 67.36 |
| FATPP | -23.71 | **-23.01** | **-40.99** | -39.68 | 26.29 | **-76.91** | -157.74 | 67.65 |



(a) In-homo



(b) Inhibit



(c) In&Ex

Figure 3: Estimated intensity function (summed over types for clarity) on synthetic datasets.

We also use three real-world datasets: **1) Stock** contains daily stock prices of 31 companies. We extract three types of event: 'up', 'down', 'unchanged' from each stock. **2) eCommerce** contains users' behavior data from a cosmetics online store. Each user's behaviors are categorized into four types: 'view', 'cart', 'remove-from-cart', 'purchase'. **3) Neuron** contains spike record of 219 M1 neurons. Every time a neuron spikes is recorded as an event, these events form a univariate event sequence. Details about synthetic and real-world datasets are reported in the Appendix B.

We compare our FATPP with RNN-based RMTPP (Du et al., 2016), Attention-based THP (Zuo et al., 2020) and SAHP (Zhang et al., 2020), intensity free models LogNormMix (Shchur et al., 2020) and AMDN (Sharma et al., 2021). In the context of clustering, our goal is to model observed data. Therefore, we do not split test set and train models on a whole dataset. The average log likelihood is reported in Table 1. We can see that our FATPP ranks top 1 in 3 datasets out of 8, ranks top 2 in 6 datasets. This means our FATPP is flexible enough to model various diverse processes and can further benefit our NTPP-MIX framework. We plot the modeled and true intensity in Fig 3. The result shows that intensity of FATPP is very close to the ground truth and it is the only NTPP among the four that can model the smooth change (Fig 3 (a)). We also do train-valid-test split to evaluate the performance for modeling unobserved data, the result is shown in Appendix D.

## 5.2 Experiments on NTPP-MIX for event sequence clustering

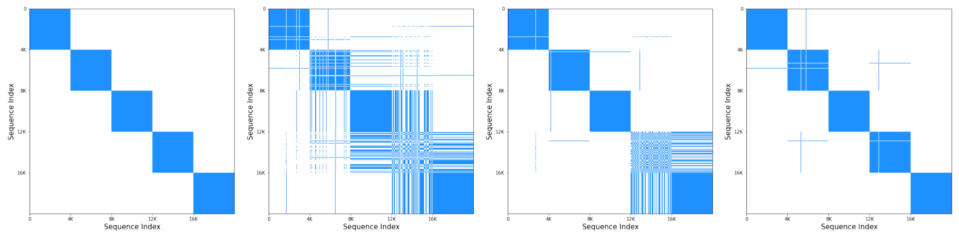
### 5.2.1 Protocols

We compare performance of the following clustering methods on synthetic and real-world datasets:

**1) HKS+BGM.** Learn a specific Hawkes process for each event sequence and apply Bayesian Gaussian Mixture (BGM) model to the learned parameters for clustering. **2) ADM4+BGM**. Learn a specific ADM4 (Zhou et al., 2013) for each event sequence and apply BGM to the learned parameters. **3) NPHC+BGM**. Learn a specific Non Parametric Hawkes Cumulant (NPHC) (Xu et al., 2016) for each event sequence and apply BGM to the learned parameters. **4) DIS+SC**. Define the distance for event sequence (Iwayama et al., 2017) and apply Spectral Clustering to the distance matrix of all pairs of sequences. **5) DMHP** (Xu & Zha, 2017). This is the most related method to our work, which mixes several Hawkes processes together to generate a Dirichlet Mixture Model of Hawkes Processes. **6) RMTPP-MIX**. Our NTPP-MIX with RMTPP (Du et al., 2016) as the NTPP component. **7) SAHP-MIX**. Our NTPP-MIX with SAHP (Zhang et al., 2020) as the NTPP component. **8) FATPP-MIX**. Our NTPP-MIX with FATPP we proposed in Sec. 4 as the NTPP component.

The first four methods can be viewed as two-step pipelines which learn features of sequences and use the extracted features for clustering. DMHP is a one-step joint model similar to our NTPP-MIX which performs clustering directly. The difference is that the mixture components are Hawkes in DMHP but NTPPs in our method. Detailed implementation of these models is shown in Appendix F.

We use the following three metrics to evaluate performance of different methods (the first two for synthetic datasets, the third for real-world datasets): **1) Clustering Purity (CP)** computes the frequency of correct assignments. **2) Adjusted Rand index (ARI)** measures the similarity of learned and ground truth assignments. **3) Log Likelihood (LL)** measures the goodness of fit for the observed event sequences. Formal definition of these metrics can be found in Appendix E.



(a) Ground Truth (b) DMHP (c) RMTPP-MIX (d) FATPP-MIX

Figure 4: Binary similarity matrix of different models on synthetic dataset $K^* = 5$ (Homo,In-homo,Inhibit,Excite,In&Ex). Element $A_{ij}$ is in blue if sequence $i$ and $j$ are in the same cluster.

Table 2: CP and ARI estimation on synthetic datasets. For each model (except DIS+SC as it is very stable), we run 5 trials with random initialization and report the mean and standard deviation.

Clustering Purity (CP): the higher the better

| $K^*$ | HKS+BGM | ADM4+BGM | NPHC+BGM | DIS+SC | DMHP | RMTPP-MIX | SAHP-MIX | FATPP-MIX |
|---|---|---|---|---|---|---|---|---|
| 2 | 0.5818 ±0.0805 | 0.6476 ±0.0812 | 0.5207 ±0.0164 | 0.9534 — | 0.9598 ±0.0007 | **0.9898** ±0.0004 | **0.9898** ±0.0019 | 0.9886 ±0.0008 |
| 3 | 0.4892 ±0.0261 | 0.4567 ±0.0131 | 0.4384 ±0.0058 | 0.9407 — | 0.8789 ±0.1069 | **0.9930** ±0.0006 | 0.9891 ±0.0008 | 0.9916 ±0.0023 |
| 4 | 0.4207 ±0.0320 | 0.3896 ±0.0135 | 0.3517 ±0.0146 | 0.2557 — | 0.5292 ±0.0413 | 0.9532 ±0.0656 | 0.9840 ±0.0008 | **0.9844** ±0.0023 |
| 5 | 0.4795 ±0.0037 | 0.4801 ±0.0064 | 0.3222 ±0.0037 | 0.2225 — | 0.6624 ±0.0905 | 0.9273 ±0.0748 | 0.9686 ±0.0109 | **0.9834** ±0.0037 |

Adjusted Rand Index (ARI): the higher the better

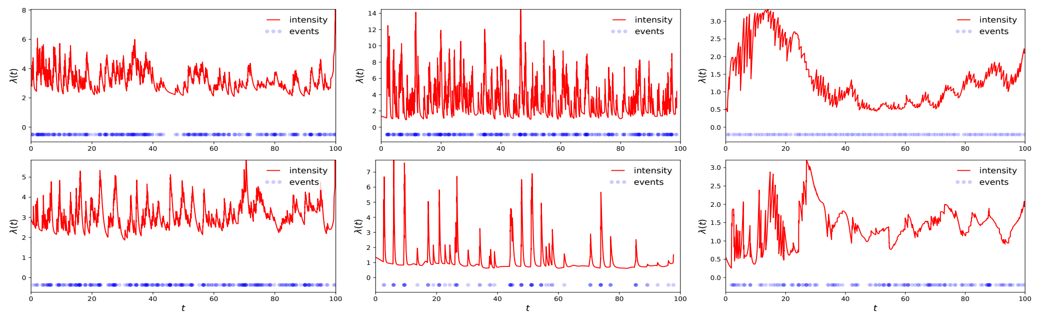
| $K^*$ | HKS+BGM | ADM4+BGM | NPHC+BGM | DIS+SC | DMHP | RMTPP-MIX | SAHP-MIX | FATPP-MIX |
|---|---|---|---|---|---|---|---|---|
| 2 | 0.0526 ±0.0852 | 0.1134 ±0.0851 | 0.0027 ±0.0023 | 0.8222 — | 0.8455 ±0.0026 | **0.9597** ±0.0015 | 0.9594 ±0.0073 | 0.9550 ±0.0032 |
| 3 | 0.0893 ±0.0424 | 0.0544 ±0.0128 | 0.0378 ±0.0044 | 0.8333 — | 0.7473 ±0.1327 | **0.9793** ±0.0018 | 0.9677 ±0.0023 | 0.9752 ±0.0068 |
| 4 | 0.0787 ±0.0258 | 0.0465 ±0.0074 | 0.0384 ±0.0108 | 0.0001 — | 0.2730 ±0.0542 | 0.9079 ±0.1106 | 0.9578 ±0.0020 | **0.9591** ±0.0059 |
| 5 | 0.1923 ±0.0331 | 0.1979 ±0.0209 | 0.0441 ±0.0023 | 0.0018 — | 0.4753 ±0.1142 | 0.8844 ±0.0885 | 0.9248 ±0.0241 | **0.9593** ±0.0037 |

#### 5.2.2 Experiments on Synthetic Datasets

We conduct experiments on four synthetic datasets with different ground truth cluster number $K^*$. For $K^* = 2$, we combine Homo and In-homo in Sec 5.1. For $K^* = 3$, we add Inhibit. For $K^* = 4$, we add Excite. For $K^* = 5$, we add In&Ex.

We first assume the ground truth cluster number $K^*$ is given and assign $K = K^*$ for each method. Table 2 shows the CP and ARI of different methods on synthetic datasets. In general, one-step joint models (DMHP and NTPP-MIX) outperform two-step pipeline models. Two-step models decompose clustering task into feature extraction and clustering the extracted feature. As these two steps are relatively independent, the extracted feature may not be suitable for clustering. Moreover, the first three models learn a unique process for each individual sequence, which may bring the risk of overfitting. Performance of DIS+SC is satisfactory on $K^* = 2, 3$ but drops sharply on $K^* = 4, 5$. This shows it is hard to define a universal distance for multi-typed event sequences (Wu et al., 2019).

For two joint models, our NTPP-MIX constantly outperforms DMHP, no matter what NTPP we use. The mixture components in DMHP are Hawkes processes, i.e. different clusters are defined as Hawkes processes with different parameters. This limits model capacity as dynamic behind data can be much more complex than Hawkes processes. Moreover, when we add sequences generated by Hawkes processes in $K^* = 4$, performance of DMHP drops suddenly and the clustering result collapses to one cluster. The mixture components of our NTPP-MIX are neural-network-based TPP models, which are more general and flexible. Among three NTPP-MIXs, our FATPP-MIX performs better when $K^* = 4, 5$, i.e. the dataset is complex. The standard deviation also shows our NTPP-MIX is more stable than DMHP. We further visualize some clustering results on $K^* = 5$ in Fig. 4. It can be seen that result of our FATPP-MIX is the closest to ground truth, while DMHP fails to separate cluster #4 out of #3 and #5, RMTPP-MIX assigns some sequences in cluster #4 to #5.



(a) Cluster #1 (b) Cluster #2 (c) Cluster #3

Figure 5: Clustered sequences and corresponding intensities of dataset Neuron using FATPP-MIX.

Table 3: LL estimation on three real-world datasets. Note that NPHC+BGM and DIS+SC are likelihood-free methods. Complex time intervals in eCommerce makes RMTPP-MIX crash.

| Dataset | HKS+BGM | ADM4+BGM | DMHP | RMTPP-MIX | SAHP-MIX | FATPP-MIX |
|---|---|---|---|---|---|---|
| Stock | -109.03 | -114.38 | -102.19 | -96.74 | -90.93 | **-76.51** |
| eCommerce | -155.37 | -127.61 | -157.70 | N/A | **-105.43** | -122.61 |
| Neuron | 51.08 | 39.21 | 30.29 | 37.85 | 68.97 | **70.44** |

We further compare NTPP-MIX's ability to choose the appropriate number of clusters with DMHP. We set the initial cluster number $K = 2K^*$, and run 10 trials with different random initialization, the selected number of clusters is shown in Table 4. We can find that number of clusters selected by our NTPP-MIX are closer to the ground truth and more concentrated (except on $K^* = 3$).

Table 4: Number of clusters selected by DMHP and NTPP-MIX on synthetic datasets. For each model, we run 10 trials.

| Ground Truth $K^*$ | 2 | 3 | 4 | 5 |
|---|---|---|---|---|
| DMHP | 3.5±0.50 | 4.9±0.30 | 2.8±0.75 | 4.4±0.92 |
| RMTPP-MIX | **2.4**±0.49 | **3.2**±0.40 | 4.3±0.46 | 5.0±0.78 |
| SAHP-MIX | 2.5±0.50 | 3.3±0.46 | **4.0**±0.00 | 6.0±0.45 |
| FATPP-MIX | 2.9±0.30 | 3.7±0.78 | 4.6±0.66 | **5.0**±0.00 |

### 5.2.3 Experiments on Real-world Datasets

We evaluate our NTPP-MIX on the three real-world datasets in Sec 5.1. Table 3 shows the performance of various methods. We can find that SAHP-MIX and FATPP-MIX outperform other methods by a large margin on all three datasets. Two-step models perform poorly and are not as interpretable as joint models: it is hard to explain what we get by clustering parameters of individual Hawkes processes in Euclidean space. DMHP performs not very well on real-world datasets. The main reason may be that the dynamics of real-world data can be much more complex than the assumption of DMHP. Moreover, DMHP collapses on Neuron and assigns all sequences to the same cluster. Compared with Table 1, we can see that the LL of SAHP-MIX and FATPP-MIX gets greater on all three datasets than using a single NTPP, this demonstrates the necessity of using mixture model for event sequence modeling. Fig. 5 shows some clustered sequences and corresponding intensities of Neuron using FATPP-MIX. We can see that in Cluster #1, the intensity has a large constant base. When an event occurs, it increases first then decreases slowly like traditional Hawkes process. In Cluster #2, the intensity has a small constant base and spikes sharply when an event occurs. In Cluster #3, the base intensity is not constant and changes with time. Occurrences of events have little influence.

Following (Xu & Zha, 2017), we also evaluate the Clustering Stability (see its formal definition in Appendix E) of DMHP and FATPP-MIX. The result is shown in Table 5. FATPP-MIX significantly outperforms DMHP which mixes Hawkes processes, showing our model is more stable and has stronger ability to discover underlying data distribution.

Table 5: Clustering stability estimation of two joint models on real-world datasets. Note that DMHP collapses on Neuron.

| Methods | Stock | eCommerce | Neuron |
|---|---|---|---|
| DMHP | 0.5347 | 0.3544 | — |
| FATPP-MIX | **0.6673** | **0.4819** | **0.6401** |

## 6 Conclusion

In this paper, we have proposed NTPP-MIX, a general framework that can utilize many existing NTPPs for event sequence clustering. We combine variational EM algorithm and SGD to efficiently train it. To further promote NTPP-MIX, we propose FATPP, a general data-driven NTPP based on attention mechanism. Experiments show the effectiveness of our NTPP-MIX (especially with FATPP) against state-of-the-arts on both synthetic and real-world datasets.

## REFERENCES

Christopher M. Bishop and Nasser M. Nasrabadi. *Pattern Recognition and Machine Learning*. J. Electronic Imaging, 2007.

Ricky T. Q. Chen, Brandon Amos, and Maximilian Nickel. Neural spatio-temporal point processes. In *International Conference on Learning Representations*, 2021.

Tian Qi Chen, Yulia Rubanova, Jesse Bettencourt, and David Kristjanson Duvenaud. Neural ordinary differential equations. In *Advances in Neural Information Processing Systems*, 2018.

D.J. Daley and Vere-Jones David. *An Introduction to the Theory of Point Processes: Volume II: General Theory and Structure*. Springer Science & Business Media, 2007.

Nan Du, H. Dai, Rakshit S. Trivedi, U. Upadhyay, M. Gomez-Rodriguez, and Le Song. Recurrent marked temporal point processes: Embedding event history to vector. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2016.

Joseph Enguehard, Dan Busbridge, Adam James Bozson, Claire Woodcock, and Nils Y. Hammerla. Neural temporal point processes for modelling electronic health records. In *Machine Learning for Health,*, 2020.

Anna Goldenberg. A survey of statistical network models. *Foundations and Trends® in Machine Learning*, 2009.

Yulong Gu. Attentive neural point processes for event forecasting. In *AAAI Conference on Artificial Intelligence*, 2021.

Alan G. Hawkes. Spectra of some self-exciting and mutually exciting point processes. *Biometrika*, 1971.

Koji Iwayama, Yoshito Hirata, and Kazuyuki Aihara. Definition of distance for nonlinear time series analysis of marked point process data. *Physics Letters A*, 2017.

Junteng Jia and Austin R Benson. Neural jump stochastic differential equations. In *Advances in Neural Information Processing Systems*, 2019.

Shuang Li, Shuai Xiao, Shixiang Zhu, Nan Du, Yao Xie, and Le Song. Learning temporal point processes via reinforcement learning. In *Advances in Neural Information Processing Systems*, 2018.

Hongyuan Mei and Jason Eisner. The neural hawkes process: A neurally self-modulating multivariate point process. In *Advances in Neural Information Processing Systems*, 2016.

Takahiro Omi, naonori ueda, and Kazuyuki Aihara. Fully neural network based model for general temporal point processes. In *Advances in Neural Information Processing Systems*, 2019.

Manfred Opper and David Saad. *Advanced mean field methods: theory and practice*. MIT press, 2001.

Karishma Sharma, Yizhou Zhang, Emilio Ferrara, and Yan Liu. Identifying coordinated accounts on social media through hidden influence and group behaviours. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2021.

Oleksandr Shchur, Marin Biloš, and Stephan Günnemann. Intensity-free learning of temporal point processes. In *International Conference on Learning Representations*, 2020.

Oleksandr Shchur, Ali Caner Türkmen, Tim Januschowski, and Stephan Günnemann. Neural temporal point processes: A review. In *International Joint Conference on Artificial Intelligence*, 2021.

Ilya Sutskever, James Martens, George E. Dahl, and Geoffrey E. Hinton. On the importance of initialization and momentum in deep learning. In *International Conference on Machine Learning*, 2013.

U. Upadhyay, Abir De, and Manuel Gomez-Rodriguez. Deep reinforcement learning of marked temporal point processes. In *Advances in Neural Information Processing Systems*, 2018.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, 2017.

Qitian Wu, Zixuan Zhang, Xiaofeng Gao, Junchi Yan, and Guihai Chen. Learning latent process from high-dimensional event sequences via efficient sampling. In *Advances in Neural Information Processing Systems*, 2019.

Weichang Wu, Junchi Yan, Xiaokang Yang, and Hongyuan Zha. Discovering temporal patterns for event sequence clustering via policy mixture model. *IEEE Transactions on Knowledge and Data Engineering*, 2020.

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

Hongteng Xu and Hongyuan Zha. A dirichlet mixture model of hawkes processes for event sequence clustering. In *Advances in Neural Information Processing Systems*, 2017.

Hongteng Xu, Mehrdad Farajtabar, and Hongyuan Zha. Learning granger causality for hawkes processes. In *International Conference on Machine Learning*, 2016.

Lizhen Xu, Jason A. Duan, and Andrew Whinston. Path to purchase: A mutually exciting point process model for online advertising and conversion. *Management Science*, 2014.

Junchi Yan, Xin Liu, Liangliang Shi, Changsheng Li, and Hongyuan Zha. Improving maximum likelihood estimation of temporal point process via discriminative and adversarial learning. In *International Joint Conference on Artificial Intelligence*, 2018.

Qiang Zhang, Aldo Lipani, Omer Kirnap, and Emine Yilmaz. Self-attentive hawkes process. In *International Conference on Machine Learning*, 2020.

K. Zhou, H. Zha, and L. Song. Learning social infectivity in sparse low-rank networks using multi-dimensional hawkes processes. In *Artificial Intelligence and Statistics*, 2013.

Simiao Zuo, Haoming Jiang, Zichong Li, Tuo Zhao, and Hongyuan Zha. Transformer hawkes process. In *International Conference on Machine Learning*, 2020.

## APPENDIX

## A CORRECTNESS AND CONVERGENCE PROOF OF TRAINING ALGORITHM

Due to page limitation, we present the derivation of our training algorithm here. Note that our model is a semi-Bayesian model where $\boldsymbol{\pi}$ and $\mathbf{Z}$ are given prior distribution but $\theta$ is not.

We first prove the correctness of Equation 6:

$$\begin{aligned}
\log p(\mathbf{S}|\theta) &= \iint q(\mathbf{Z}, \boldsymbol{\pi}) \log p(\mathbf{S}|\theta) d\mathbf{Z} d\boldsymbol{\pi} \\
&= \iint q(\mathbf{Z}, \boldsymbol{\pi}) \log \frac{p(\mathbf{S}|\theta) p(\mathbf{Z}, \boldsymbol{\pi}|\mathbf{S}, \theta)}{p(\mathbf{Z}, \boldsymbol{\pi}|\mathbf{S}, \theta)} d\mathbf{Z} d\boldsymbol{\pi} \\
&= \iint q(\mathbf{Z}, \boldsymbol{\pi}) \log \frac{p(\mathbf{S}, \mathbf{Z}, \boldsymbol{\pi}|\theta)}{p(\mathbf{Z}, \boldsymbol{\pi}|\mathbf{S}, \theta)} d\mathbf{Z} d\boldsymbol{\pi} \\
&= \iint q(\mathbf{Z}, \boldsymbol{\pi}) \log \frac{p(\mathbf{S}, \mathbf{Z}, \boldsymbol{\pi}|\theta)}{q(\mathbf{Z}, \boldsymbol{\pi})} \frac{q(\mathbf{Z}, \boldsymbol{\pi})}{p(\mathbf{Z}, \boldsymbol{\pi}|\mathbf{S}, \theta)} d\mathbf{Z} d\boldsymbol{\pi} \\
&= \iint q(\mathbf{Z}, \boldsymbol{\pi}) \log \frac{p(\mathbf{S}, \mathbf{Z}, \boldsymbol{\pi}|\theta)}{q(\mathbf{Z}, \boldsymbol{\pi})} d\mathbf{Z} d\boldsymbol{\pi} + \iint q(\mathbf{Z}, \boldsymbol{\pi}) \log \frac{q(\mathbf{Z}, \boldsymbol{\pi})}{p(\mathbf{Z}, \boldsymbol{\pi}|\mathbf{S}, \theta)} d\mathbf{Z} d\boldsymbol{\pi} \\
&= \mathcal{L}(q, \theta) + \mathrm{KL}(q \| p)
\end{aligned} \tag{19}$$

Move $\mathrm{KL}(q\|p)$ to the left side, we can get Equation 6. Equation 6 tells us that maximizing $\mathcal{L}(q, \theta)$ equals to maximizing $\log p(\mathbf{S}|\theta)$ and minimizing $\mathrm{KL}(q\|p)$. Maximizing $\log p(\mathbf{S}|\theta)$ is our original optimization goal. Minimizing $\mathrm{KL}(q\|p)$ gives us a good approximation of $p(\mathbf{Z}, \boldsymbol{\pi}|\mathbf{S}, \theta)$ which is the posterior of cluster assignment. Assuming $q(\mathbf{Z}, \boldsymbol{\pi}) = q(\mathbf{Z})q(\boldsymbol{\pi})$, our goal changes to maximizing $\mathcal{L}(q, \theta)$ with respect to $q(\mathbf{Z}), q(\boldsymbol{\pi})$ and $\theta$.

We now derive and prove the convergence of our training algorithm. We maximize $\mathcal{L}(q, \theta)$ with respect to $q(\mathbf{Z}), q(\boldsymbol{\pi}), \theta$ iteratively. First, we fix $q(\boldsymbol{\pi}), \theta$ and update $q(\mathbf{Z})$, thus terms irrelevant to $q(\mathbf{Z})$ can be viewed as constant:

$$\begin{aligned}
\mathcal{L}(q, \theta) =& \mathcal{L}(q(\mathbf{Z}), q(\boldsymbol{\pi}), \theta) \\
=& \iint q(\mathbf{Z}) q(\boldsymbol{\pi}) \log \frac{p(\mathbf{S}, \mathbf{Z}, \boldsymbol{\pi}|\theta)}{q(\mathbf{Z}) q(\boldsymbol{\pi})} d\mathbf{Z} d\boldsymbol{\pi} \\
=& \int q(\mathbf{Z}) \int q(\boldsymbol{\pi}) \log p(\mathbf{S}, \mathbf{Z}, \boldsymbol{\pi}|\theta) d\boldsymbol{\pi} d\mathbf{Z} - \int q(\boldsymbol{\pi}) \int q(\mathbf{Z}) \log q(\mathbf{Z}) d\mathbf{Z} d\boldsymbol{\pi} \\
& - \iint q(\mathbf{Z}) q(\boldsymbol{\pi}) \log q(\boldsymbol{\pi}) d\mathbf{Z} d\boldsymbol{\pi} \\
=& \int q(\mathbf{Z}) \mathbb{E}_{q(\boldsymbol{\pi})}[\log p(\mathbf{S}, \mathbf{Z}, \boldsymbol{\pi}|\theta)] d\mathbf{Z} - \int q(\mathbf{Z}) \log q(\mathbf{Z}) d\mathbf{Z} + \mathrm{Const} \\
=& - \mathrm{KL}[q(\mathbf{Z}) \| \widetilde{p}(\mathbf{S}, \mathbf{Z}|\theta)] + \mathrm{Const}
\end{aligned} \tag{20}$$

where we define a new distribution with $\log \widetilde{p}(\mathbf{S}, \mathbf{Z}|\theta) = \mathbb{E}_{q(\boldsymbol{\pi})}[\log p(\mathbf{S}, \mathbf{Z}, \boldsymbol{\pi}|\theta)]$, as $\mathrm{KL}[q(\mathbf{Z}) \| \widetilde{p}(\mathbf{S}, \mathbf{Z}|\theta)] \geq 0$ and the minimum occurs when $q(\mathbf{Z}) = \widetilde{p}(\mathbf{S}, \mathbf{Z}|\theta)$. Thus in the $t$ round, setting $\log q^{(t)}(\mathbf{Z}) = \mathbb{E}_{q^{(t-1)}(\boldsymbol{\pi})}[\log p(\mathbf{S}, \mathbf{Z}, \boldsymbol{\pi}|\theta^{(t-1)})] + \mathrm{Const}$ (additive constant is set for normalization), we can get:

$$\mathcal{L}(q^{(t)}(\mathbf{Z}), q^{(t-1)}(\boldsymbol{\pi}), \theta^{(t-1)}) \geq \mathcal{L}(q^{(t-1)}(\mathbf{Z}), q^{(t-1)}(\boldsymbol{\pi}), \theta^{(t-1)}) \tag{21}$$

Then, we fix $q(\mathbf{Z}), \theta$ and update $q(\boldsymbol{\pi})$, thus terms irrelevant to $q(\boldsymbol{\pi})$ can be viewed as constant. Similarly, we can get:

$$\begin{aligned}\mathcal{L}(q,\theta) =&\mathcal{L}(q(\mathbf{Z}), q(\boldsymbol{\pi}), \theta)\\ =&\iint q(\mathbf{Z})q(\boldsymbol{\pi})\log\frac{p(\mathbf{S},\mathbf{Z},\boldsymbol{\pi}|\theta)}{q(\mathbf{Z})q(\boldsymbol{\pi})}d\mathbf{Z}d\boldsymbol{\pi}\\ =&\int q(\boldsymbol{\pi})\int q(\mathbf{Z})\log p(\mathbf{S},\mathbf{Z},\boldsymbol{\pi}|\theta)d\mathbf{Z}d\boldsymbol{\pi} - \int q(\mathbf{Z})\int q(\boldsymbol{\pi})\log q(\boldsymbol{\pi})d\boldsymbol{\pi}d\mathbf{Z}\\ &-\iint q(\mathbf{Z})q(\boldsymbol{\pi})\log q(\mathbf{Z})d\mathbf{Z}d\boldsymbol{\pi}\\ =&\int q(\boldsymbol{\pi})\mathbb{E}_{q(\mathbf{Z})}[\log p(\mathbf{S},\mathbf{Z},\boldsymbol{\pi}|\theta)]d\boldsymbol{\pi} - \int q(\boldsymbol{\pi})\log q(\boldsymbol{\pi})d\boldsymbol{\pi} + \text{Const}\\ =&-\text{KL}[q(\boldsymbol{\pi})\|\widetilde{p}(\mathbf{S},\boldsymbol{\pi}|\theta)] + \text{Const}\end{aligned} \tag{22}$$

where $\log\widetilde{p}(\mathbf{S},\boldsymbol{\pi}|\theta) = \mathbb{E}_{q(\mathbf{Z})}[\log p(\mathbf{S},\mathbf{Z},\boldsymbol{\pi}|\theta)]$, as $\text{KL}[q(\boldsymbol{\pi})\|\widetilde{p}(\mathbf{S},\boldsymbol{\pi}|\theta)] \geq 0$ and the minimum occurs when $q(\boldsymbol{\pi}) = \widetilde{p}(\mathbf{S},\boldsymbol{\pi}|\theta)$. Thus in the $t$-th round, setting $\log q^{(t)}(\boldsymbol{\pi}) = \mathbb{E}_{q^{(t)}(\mathbf{Z})}[\log p(\mathbf{S},\mathbf{Z},\boldsymbol{\pi}|\theta^{(t-1)})] + \text{Const}$, we can obtain:

$$\mathcal{L}(q^{(t)}(\mathbf{Z}), q^{(t)}(\boldsymbol{\pi}), \theta^{(t-1)}) \geq \mathcal{L}(q^{(t)}(\mathbf{Z}), q^{(t-1)}(\boldsymbol{\pi}), \theta^{(t-1)}) \tag{23}$$

Finally, we fix $q(\mathbf{Z}), q(\boldsymbol{\pi})$ and update $\theta$, thus terms irrelevant to $\theta$ can be viewed as constant. As $\theta$ are parameters for neural networks, it is hard to directly get an optimal form like $q(\mathbf{Z}), q(\boldsymbol{\pi})$, we use SGD for approximation:

$$\begin{aligned}&\mathcal{L}(q,\theta)\\ =&\mathcal{L}(q(\mathbf{Z}), q(\boldsymbol{\pi}), \theta)\\ =&\iint q(\mathbf{Z})q(\boldsymbol{\pi})\log\frac{p(\mathbf{S},\mathbf{Z},\boldsymbol{\pi}|\theta)}{q(\mathbf{Z})q(\boldsymbol{\pi})}d\mathbf{Z}d\boldsymbol{\pi}\\ =&\iint q(\mathbf{Z})q(\boldsymbol{\pi})\log p(\mathbf{S},\mathbf{Z},\boldsymbol{\pi}|\theta)d\mathbf{Z}d\boldsymbol{\pi} - \iint q(\mathbf{Z})q(\boldsymbol{\pi})\log\left(q(\mathbf{Z})q(\boldsymbol{\pi})\right)d\mathbf{Z}d\boldsymbol{\pi}\\ =&\iint q(\mathbf{Z})q(\boldsymbol{\pi})\log\left(p(\boldsymbol{\pi})p(\mathbf{Z}|\boldsymbol{\pi})p(\mathbf{S}|\mathbf{Z},\theta)\right)d\mathbf{Z}d\boldsymbol{\pi} + \text{Const}\\ =&\iint q(\mathbf{Z})q(\boldsymbol{\pi})\log p(\mathbf{S}|\mathbf{Z},\theta)d\mathbf{Z}d\boldsymbol{\pi} + \iint q(\mathbf{Z})q(\boldsymbol{\pi})\log\left(p(\boldsymbol{\pi})p(\mathbf{Z}|\boldsymbol{\pi})\right)d\mathbf{Z}d\boldsymbol{\pi} + \text{Const}\\ =&\int q(\mathbf{Z})\log p(\mathbf{S}|\mathbf{Z},\theta)d\boldsymbol{\pi} + \text{Const}\\ =&\underbrace{\mathbb{E}_{q(\mathbf{Z})}[\log p(\mathbf{S}|\mathbf{Z},\theta)]}_{\mathcal{L}(\theta)} + \text{Const}\end{aligned} \tag{24}$$

We set $\theta^{(t)} = \text{SGD}(-\mathcal{L}(\theta))$. However, as the optimization problem is non-convex, we can not guarantee $\mathcal{L}$ always gets greater after one-step in SGD. To address this problem, in each M-step, we perform $M > 1$ SGD steps for optimization. Therefore, it is reasonable to assume that after $M$ steps:

$$\mathcal{L}(q^{(t)}(\mathbf{Z}), q^{(t)}(\boldsymbol{\pi}), \theta^{(t)}) \geq \mathcal{L}(q^{(t)}(\mathbf{Z}), q^{(t)}(\boldsymbol{\pi}), \theta^{(t-1)}) \tag{25}$$

Summarize Equation 21, 23 and 25, we can get:

$$\mathcal{L}(q^{(t)}(\mathbf{Z}), q^{(t)}(\boldsymbol{\pi}), \theta^{(t)}) \geq \mathcal{L}(q^{(t-1)}(\mathbf{Z}), q^{(t-1)}(\boldsymbol{\pi}), \theta^{(t-1)}) \tag{26}$$

This suggests that after each round, the objective function $\mathcal{L}(q,\theta)$ gets greater. Perform these three steps iteratively, we can get the optimal solution. In practice, we also use mini-batch training, details can be find in Algorithm 1.

## B Details of Datasets

### B.1 Synthetic Datasets

For synthetic datasets, we use five different processes to generate sequences. The corresponding intensity functions are as follows:
**1) Homogeneous Possion**:

$$\lambda(t, u|\mathcal{H}_t) = 0.5u$$

**2) Inhomogeneous Possion**:

$$\lambda(t, u|\mathcal{H}_t) = 1.5 + \sin \frac{u\pi t}{3}$$

**3) Inhibition**:

$$\lambda(t, u|\mathcal{H}_t) = \max\left\{0, \mu_u + \sum_{t_i<t} \alpha_{uu_i} \exp\left(-\beta(t - t_i)\right)\right\}$$
$$\mu_u \sim \text{Uniform}(2, 4) \quad \alpha_{uu_i} \sim \text{Uniform}(-1, 0) \quad \beta \sim \text{Uniform}(1, 3)$$

note that $\alpha_{uu_i} \leq 0$, thus the occurrence of a new event will cause the intensity to decrease.
**4) Excitation (Hawkes)**:

$$\lambda(t, u|\mathcal{H}_t) = \mu_u + \sum_{t_i<t} \alpha_{uu_i} \exp\left(-\beta(t - t_i)\right)$$
$$\mu_u \sim \text{Uniform}(0.2, 0.4) \quad \alpha_{uu_i} \sim \text{Uniform}(0, 1) \quad \beta \sim \text{Uniform}(1, 3)$$

note that $\alpha_{uu_i} \geq 0$, thus the occurrence of a new event will cause the intensity to increase.
**5) Excitation & Inhibition**:

$$\lambda(t, u|\mathcal{H}_t) = \max\left\{0, \mu_u + \sum_{t_i<t} \alpha_{uu_i} \exp\left(-\beta(t - t_i)\right)\right\}$$
$$\mu_u \sim \text{Uniform}(1, 2) \quad \alpha_{uu_i} \sim \text{Uniform}(-1, 1) \quad \beta \sim \text{Uniform}(1, 2)$$

note that $\alpha_{uu_i}$ can be negative or positive, thus relation between two type of events is either inhibition or excitation.

For each of the above five processes, we generate 4,000 event sequences using python library **tick**[2]. Each sequence contains $L_n = 50$ events with number of event types $C = 5$.

We plot some sequences and the corresponding intensity function of the five processes in Fig 6.

### B.2 Real-world Datasets

The details of the three real-world datasets and the corresponding preprocessing methods are as follows:

**1) Stock**[3] contains daily stock prices of 31 companies from 2006 to 2017. For each stock, an 'up' event is recorded when the closing price changes by more than $+2\%$ of its opening price, an 'down' event is recorded when the closing price changes by more than $-2\%$ of its opening price, an 'unchanged' event is recorded when the closing price changes by less than $\pm 1\%$ of its opening price. We further partition sequences by every season and obtain 1,488 sequences with average length of 49.

**2) eCommerce**[4] contains users' behavior data from a cosmetics online store in December 2019. Each user's behaviors are categorized into four types: 'view', 'cart', 'remove-from-cart', 'purchase'. In each day, we rank users by the number of actions they take and select users ranked 50 to 249 to prevent the sequences from being too long or too short. This dataset contains 6,200 sequences with

[2]https://x-datainitiative.github.io/tick/index.html
[3]The dataset is available on www.kaggle.com/szrlee/stock-time-series-20050101-to-20171231.
[4]The dataset is available on www.kaggle.com/mkechinov/ecommerce-events-history-in-cosmetics-shop.

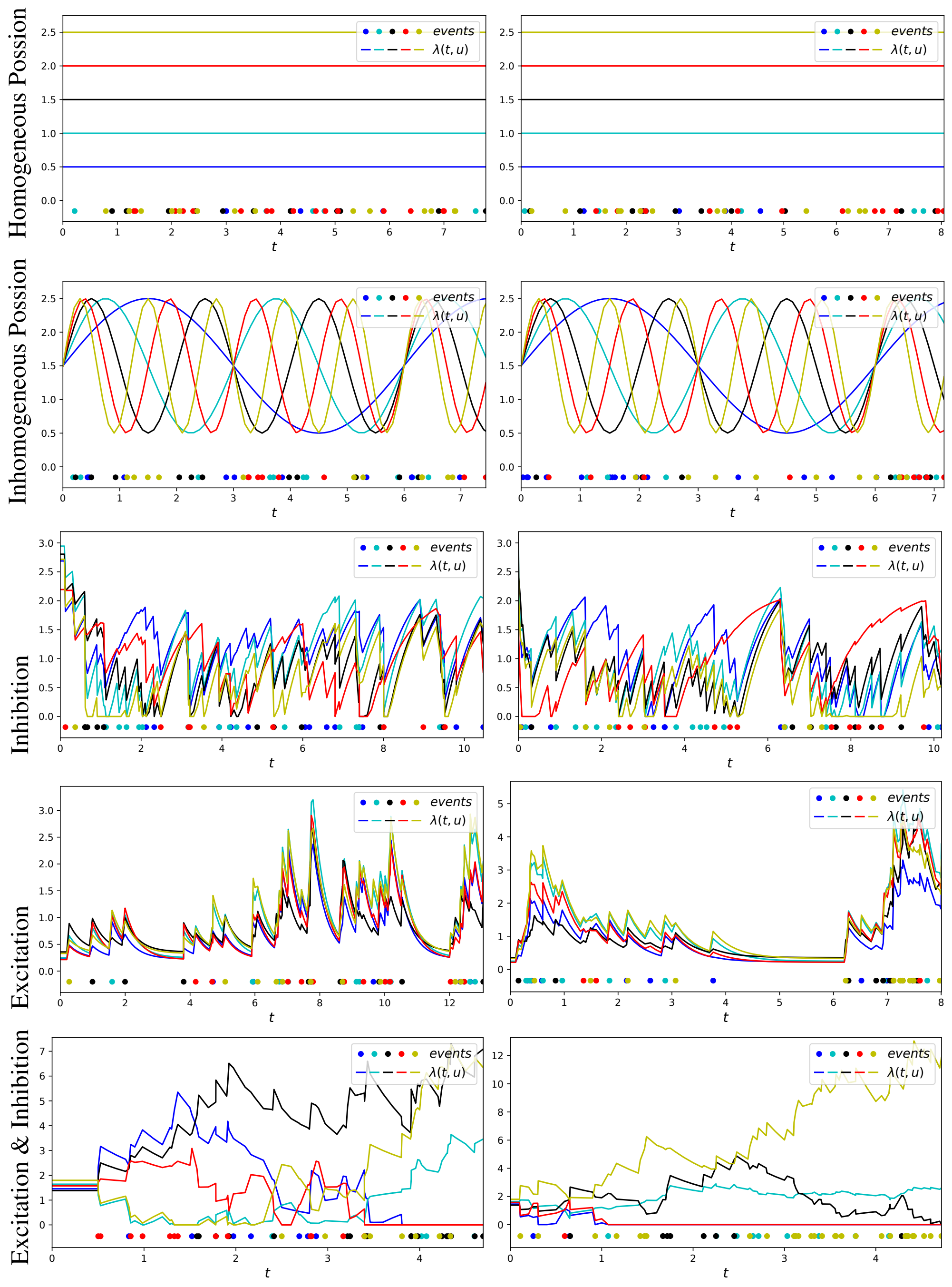


Figure 6: Some event sequences and corresponding ground truth intensities of synthetic dataset. The two sequences in each row are generated by the same process.

average length of 64.

**3) Neuron**[5] contains spike record of 219 M1 neurons. For each neuron, every time it spikes is recorded as an event, these events form a univariate event sequence. We further partition sequences by every 100ms and drop sequences with a length greater than 500. This dataset contains 3,718

[5]The dataset is available on www.kordinglab.com/spykes/index.html.

sequences with average length of 265. Note that sequences in this dataset are univariate, which are special cases of the data we study.

## C THE EFFICIENCY ANALYSIS FOR FATPP

In this section, we analyze the efficiency of FATPP compared with other Attention-based NTPP, e.g. AMDN (Sharma et al., 2021), THP (Zuo et al., 2020).

Table 6: Computation complexity of encoding, intensity formulation, overall training and minimum number of sequential operations. $L$ is the sequence length. $K$ is the number of mixture components in AMDN. $S$ is the number of Monte Carlo samples. Assume $K, S \ll L$.

| Method | Encoding | Intensity Formulation | Overall | Sequential |
|---|---|---|---|---|
| AMDN | $O(L^2)$ | $O(KL)$ | $O(L^2)$ | $O(1)$ |
| THP | $O(L^2)$ | $O(SL)$ | $O(L^2)$ | $O(1)$ |
| FATPP | $O(L^2)$ | $O(SL^2)$ | $O(SL^2)$ | $O(1)$ |

We now consider an event sequence of length $L$ and view $d$-dimensional (dimension of hidden state) vector multiplication as an elementary operation. Table 6 shows the computation complexity of the three models in different period. All three models need a forward pass of transformer to encode the events, so the encoding complexity is $O(L^2)$. For intensity formulation, for each of the $L$ time intervals, AMDN uses a mixture of $K$ log-normal distributions to model the PDF, thus its complexity is $O(KL)$. THP uses $S$ Monte Carlo samples to approximate the integration in Equation 2. The parameters in the intensity are only computed once, making each sample a $O(1)$ operation. Therefore the intensity formulation complexity of THP is $O(SL)$. FATPP also uses $S$ Monte Carlo samples to approximate the integration but each sample requires a query operation of transformer, so it's complexity is $O(SL^2)$. Jointly considering encoding and intensity formulation, assuming $K, S \ll L$, the overall complexity is $O(L^2)$ for AMDN and THP and $O(SL^2)$ for FATPP. Theoretically, all operations in these models can be parallelized, so the number of sequential operations is $O(1)$ for all three models.

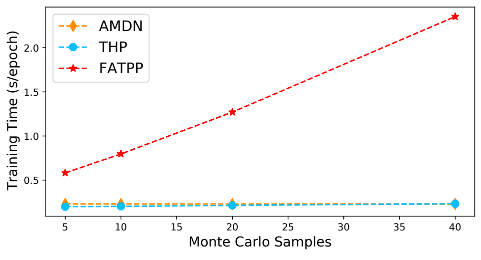
Adding a factor of $S$ to the complexity, we can model the intensity of each mark more accurately, i.e. $\lambda(t, u|\mathcal{H}_t)$ for each $u$. On the contrary, AMDN is flexible enough to model the overall intensity $\lambda(t|\mathcal{H}_t) = \sum_u \lambda(t, u|\mathcal{H}_t)$, but assumes the probabilities of time $t$ and marker $u$ are independent. In other words, given the history, the conditional probability of which kinds of events to occur remains constant. THP makes parametric assumptions about the intensity, which is not flexible enough.

We conduct experiments to confirm our above analysis. We use the Inhomogeneous Possion in Appendix B.1 to generate synthetic data as its time and mark are dependent. We first generate a dataset with 5,000 sequences to test the effect of Monte Carlo samples. We split the dataset into training, validation, and testing sets at a ratio of 6:2:2 and train each model on the training set and reports the corresponding negative log likelihood (NLL) on testing set. All models run on a single RTX-2080Ti (11GB) GPU with equal hidden dimensions, attention heads, etc. Figure 7 shows the time cost and NLL on testing set with different sample times. We can see that number of samples has little influence on THP. While the training time of FATPP increases linearly with samples. However, the NLL of FATPP is the lowest among the three models and 10 samples result in the best performance.

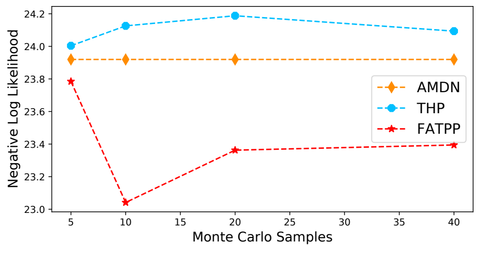
We then generate datsets with 10,000, 15,000, 20,000 sequences, split them and evaluate training time and NLL. In this experiment, we fix the sample times as 10 for THP and FATPP. The result is shown in Figure 8. The training time increases linearly with the size of dataset. While time cost of FATPP is around four times that of other models, but it constantly outperforms other two models on datasets with different size.

In general, FATPP is more time-consuming than AMDN and THP. But it models the intensity of each individual marks more accurately.

## D EXPERIMENTS ON SINGLE NTPPS FOR MODELING UNOBSERVED DATA

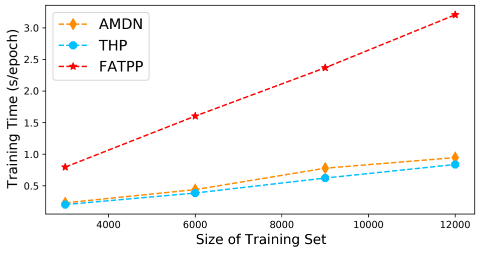


(a)

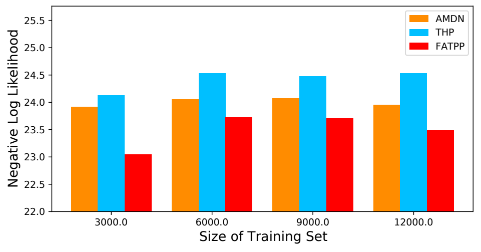


(b)

Figure 7: Results with respect to different Monte Carlo sample times. (a) Training time cost with different sample times. (b)Negative log likelihood on testing set with different sample times.



(a)



(b)

Figure 8: Results with respect to different size of dataset. (a) Training time cost with different size of dataset. (b)Negative log likelihood on testing set with different size of dataset.

In this section, we split each dataset in Sec 5.1 into training, validation, and testing sets at a ratio of 6:2:2. We train the model on training set and evaluate LL on testing set, other settings are same as in Sec 5.1.

The result is shown in Table 7. We can see that our FATPP ranks top 1 in 5 datasets out of 8, ranks top 3 in 7 datsets. Results under both observed and unobserved settings show that FATPP is able to model various diverse processes, and further benefits NTPP-MIX.

Table 7: Average log likelihood estimation of single NTPPs on testing set.

| Dataset | Homo | In-homo | Inhibit | Excite | In&Ex | Stock | eCommerce | Neuron |
|---|---|---|---|---|---|---|---|---|
| RMTPP | -23.87 | -24.65 | -44.30 | -41.35 | 25.66 | -102.12 | -168.57 | 46.00 |
| THP | -23.87 | -24.92 | -42.13 | -41.45 | **25.80** | -91.92 | N/A | 40.87 |
| SAHP | -23.88 | -24.61 | -43.77 | -40.47 | 25.71 | -94.16 | -116.04 | 58.69 |
| LogNormMix | -23.92 | -24.75 | -42.64 | -40.76 | 25.51 | -74.60 | -122.44 | **67.75** |
| AMDN | -23.99 | -25.00 | -42.32 | -40.60 | 25.56 | -73.07 | **-99.76** | 64.37 |
| FATPP | **-23.86** | **-24.36** | **-41.29** | **-40.43** | 25.69 | **-68.75** | -152.38 | 62.67 |

## E Details of Evaluation Metrics

**1) Clustering Purity (CP)**. Given the ground truth cluster assignments, CP computes the frequency of correct assignments. Formally:

$$\text{CP} = \frac{1}{N}\sum_{k=1}^{K}\max_{j\in\{1,\ldots,K^*\}}|\omega_k \cap \zeta_j| \tag{27}$$

where $\omega_k$ is the set of sequences assigned to the $k$-th cluster by the model, $\zeta_j$ is the real set of sequences belonging to the $j$-th cluster, $K$ and $K^*$ are learned and ground truth cluster number. CP lies in $[0, 1]$. Higher CP indicates better clustering, a perfect clustering has a CP of 1, a bad clustering has a CP close to 0.

**2) Adjusted Rand Index (ARI)**. Given the learned and ground truth cluster assignments, ARI measures the similarity of these two assignments. Formally:

$$\text{RI} = \frac{a+b}{\binom{N}{2}}, \quad \text{ARI} = \frac{\text{RI} - \mathbb{E}[\text{RI}]}{\max(\text{RI}) - \mathbb{E}[\text{RI}]} \tag{28}$$

where $a$ is the number of pairs that are in the same set in both learned and real assignment, $b$ is the number of pairs in different sets in both learned and real assignment, $\mathbb{E}[\text{RI}]$ is the expected RI of random assignments. The Rand Index (RI) measures the similarity, while ARI further adjusts it so that a random assignment has a negative ARI or an ARI close to 0. ARI lies in $[-1, 1]$. Higher ARI indicates better clustering, a perfect clustering has a ARI of 1.

**3) Log Likelihood (LL)**. When the ground-truth assignment is unknown, we evaluate LL:

$$\text{LL} = \frac{1}{N} \sum_{n=1}^{N} \log p(S_n | \theta_{\widetilde{\mathbf{z}_n}}) \tag{29}$$

where $\widetilde{\mathbf{z}}_n$ is the predicted cluster of $S_n$. For each sequence $S_n$, the model assigns it to cluster $\widetilde{\mathbf{z}_n}$ and use the corresponding basic model $p(\cdot|\theta_{\widetilde{\mathbf{z}}_n})$ (e.g. Hawkes for HKS+BGM and DMHP, NTPP for NTPP-MIX) to evaluate the likelihood. Higher LL indicates better clustering to a certain extent.

**4) Clustering Stability (CS)**. CS measures the consistency and stability of a model via cross-validation (Goldenberg, 2009).

In our experiments, we run $J = 10$ trails for each model. In trail $i$, we randomly select 60% sequences from the dataset and use them to train the model. Then the trained model is applied to the whole dataset to get an assignment $C_i$. As random selection somehow represents the underlying distribution of the dataset, assignments of different trail should be similar for a stable model. Then CS is defined as:

$$CS = \frac{1}{\binom{J}{2}} \sum_{i=1}^{J} \sum_{j \neq i} \text{ARI}(C_i, C_j) \tag{30}$$

CS lies in [-1, 1]. Higher CS indicates more stable clustering but does not necessarily mean a better model: assigning all sequences to one cluster will get a CS of 1. Therefore we only use CS as a supporting metric for stability comparsion.

## F Implementation details of models

**1) HKS+BGM.** Hawkes is implemented by python library **tick**. Hyper-parameter *decay* is set by grid search in $[100, 10, 1, 0.1, 0.01, 0.001]$. Other hyper-parameters are set as default values. BGM is implemented by python library **sklearn**[6], hyper-parameter *max_iter* is set to 500 and other hyper-parameters are set as default values.
**2) ADM4+BGM.** ADM4 is implemented by python library **tick**. Hyper-parameter *decay* is set by grid search in $[1000, 100, 10, 1, 0.1, 0.01, 0.001]$. Other hyper-parameters are set as default values. Implementation of BGM is same as HKS+BGM.
**3) NPHC+BGM.** NPHC is implemented by python library **tick**. Hyper-parameter *integration_support* is set to 0.2. Other hyper-parameters are set as default values. Implementation of BGM is same as HKS+BGM.
**4) DIS+SC.** DIS is implemented by the matlab toolbox **THAP** [7], all parameters are set as default. SC is implemented by python library **sklearn**.
**5) DMHP.** DMHP is implemented by the matlab toolbox **THAP**. On synthetic datasets, *outer iter* is set to 10, *inner iter* is set to 5. On real-world datasets, *outer iter* is set to 20, *inner iter* is set to 4.
**6) RMTPP-MIX.** We implement RMTPP-MIX using PyTorch. On both synthetic and real-world dataset, batch size $B$ is set to 128, training epoch $T$ is set to 100, number of SGD in M-steps is set to 5, weights for prior $\alpha_0$ is set to $\frac{1}{K}$, we pre-train each RMTPP for 5 epochs on the whole dataset. On synthetic dataset, dimension of hidden state $d$ is set to 32, $q(\boldsymbol{\pi})$ update rate $\eta$ is set to 0.01, minimum cluster size $\epsilon$ is set to $\frac{N}{5*K}$, Adam with learning rate 0.005 is used for SGD optimization; On

[6]https://scikit-learn.org/stable/index.html
[7]https://github.com/HongtengXu/Hawkes-Process-Toolkit

real-world dataset, dimension of hidden state $d$ is set to 64, $q(\boldsymbol{\pi})$ update rate $\eta$ is set to $\frac{B}{N}$, minimum cluster size $\epsilon$ is set to $\frac{N}{10*K}$, Adam with learning rate 0.0005 is used for SGD optimization.
**7) SAHP-MIX.** We implement SAHP-MIX using PyTorch. On both synthetic and real-world dataset, we use 2-head attention, encoder layer is set to 1, batch size $B$ is set to 128, training epoch $T$ is set to 100, number of SGD in M-steps is set to 5, weights for prior $\alpha_0$ is set to $\frac{1}{K}$, we pre-train each SAHP for 5 epochs on the whole dataset. On synthetic dataset, dimension of hidden state $d$ is set to 32, $q(\boldsymbol{\pi})$ update rate $\eta$ is set to 0.01, minimum cluster size $\epsilon$ is set to $\frac{N}{5*K}$, Adam with learning rate 0.005 is used for SGD optimization; On real-world dataset, dimension of hidden state $d$ is set to 64, $q(\boldsymbol{\pi})$ update rate $\eta$ is set to $\frac{B}{N}$, minimum cluster size $\epsilon$ is set to $\frac{N}{10*K}$, Adam with learning rate 0.0005 is used for SGD optimization.
**8) FATPP-MIX.** We implement FATPP-MIX using PyTorch. Hyper-parameter setting is same as SAHP-MIX except training epoch $T$ is set to 50 on synthetic datasets.

NTPP-MIX runs on a single RTX-2080Ti (11GB) GPU and other models run on Intel i9-7920X CPU @ 2.90GHz with 128GB RAM.

## G Experiments on Hyper-parameter Setting

There are some hyper-parameters in our NTPP-MIX which may affect model performance. In this section, we conduct experiments to investigate their influence and give suggestions for hyper-parameter setting. We conduct experiments on synthetic dataset $K^* = 5$ (Homo + In-homo + Inhibit + Excite + In&Ex) and use FATPP as the basic component.

### G.1 Number of SGD steps in one M-step: $M$

In Algorithm 1, we perform $M$ SGD steps in one M-step. And in Appendix A, we assume after $M$ SGD steps, $\mathcal{L}(\theta)$ gets greater so that the whole training algorithm can converge. We now conduct experiments to verify it. We use FATPP-MIX with 2-head attention, dimension of hidden state $d$ is set to 32, encoder layer is set to 1, batch size $B$ is set to 128, training epoch $T$ is set to 50, $q(\boldsymbol{\pi})$ update rate $\eta$ is set to 0.01, minimum cluster size $\epsilon$ is set to $\frac{N}{5*K} = 800$, weights for prior $\alpha_0$ is set to $\frac{1}{K}$, Adam with learning rate 0.005 is used for SGD optimization, we pre-train each RMTPP for 5 epochs on the whole dataset. We then set $M$ to $[1, 3, 5, 7]$ and evaluate the CP and ARI, result is shown in Table 8.

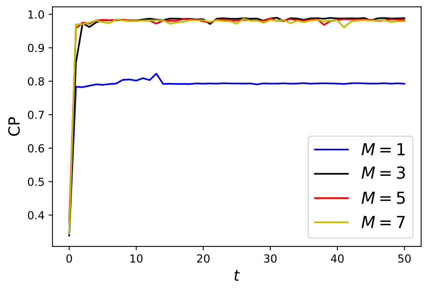
We can see that FATPP-MIX performs poorly when $M = 1$ (but still much better than baselines such as DMHP). Actually, the selected number of clusters is 4. This confirms that we need multiple SGD steps in one M-step. Setting $M = 3, 5, 7$, the model converges well and we get a series of relatively close results. Fig 9 shows CP and ARI curves during training, which also confirm the model's convergence. Considering the training speed, we recommend setting $M$ in {3, 4, 5}.

Table 8: Performance of FATPP-MIX with respect to hyper-parameter SGD steps $M$

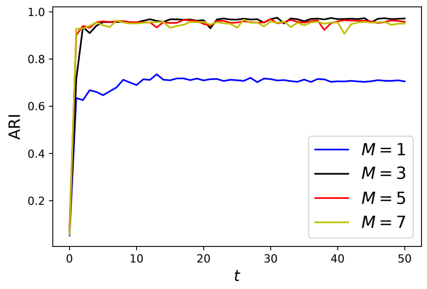
| $M$ | 1 | 3 | 5 | 7 |
|---|---|---|---|---|
| CP | 0.7927 | 0.9886 | 0.9828 | 0.9797 |
| ARI | 0.7053 | 0.9729 | 0.9577 | 0.9503 |
| Speed (seconds/epoch) | 13.33 | 37.07 | 58.77 | 85.90 |

### G.2 Batch Size: $B$

In Algorithm 1, we use mini-batch training: randomly sample $B$ sequences to represent the underlying distribution and update model based on the sampled data. Therefore, the batch size $B$ may affect model performance, we now investigate it. We use FATPP-MIX with 2-head attention, dimension of hidden state $d$ is set to 32, encoder layer is set to 1, training epoch $T$ is set to 50, number of SGD steps $M$ is set to 5, $q(\boldsymbol{\pi})$ update rate $\eta$ is set to 0.01, minimum cluster size $\epsilon$ is set to $\frac{N}{5*K} = 800$, weights for prior $\alpha_0$ is set to $\frac{1}{K}$, Adam with learning rate 0.005 is used for SGD optimization, we pre-train each RMTPP for 5 epochs on the whole dataset. We then set $B$ to $[64, 128, 256, 512]$ and evaluate the CP and ARI, result is shown in Table 9, CP and ARI curves during training are shown in Fig 10.



(a) CP



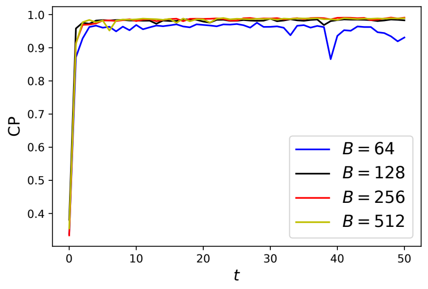
(b) ARI

Figure 9: CP and ARI curves during training process with different $M$.

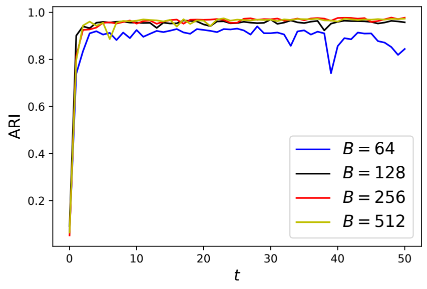
We can see that when the batch size is 64, the result falls significantly behind other settings. This is because a small batch size may result in biased samples, which can not represent underlying distribution well. When $B \geq 128$, the results are satisfactory and larger $B$ gives relatively better results. Therefore, we recommend setting B as large as possible within the reasonable range.

Table 9: Performance of FATPP-MIX with respect to batch size $B$

| $B$ | 64 | 128 | 256 | 512 |
|---|---|---|---|---|
| CP | 0.9310 | 0.9828 | 0.9908 | 0.9892 |
| ARI | 0.8444 | 0.9577 | 0.9773 | 0.9732 |



(a) CP



(b) ARI

Figure 10: CP and ARI curves during training process with different batch size $B$.

## G.3 Update rate for $q(\pi)$: $\eta$

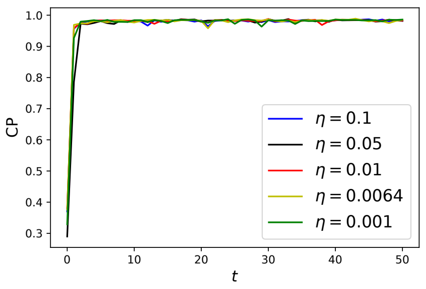
In Algorithm 1, we sample a batch of data, compute parameters for $q(\boldsymbol{\pi})$ on this batch. As new parameters are computed on part of the dataset, we introduce $\eta$ to determine how much to update. $\eta$ acts as the momentum in machine learning (Sutskever et al., 2013), we now evaluate how it affects the model.

We use FATPP-MIX with 2-head attention, dimension of hidden state $d$ is set to 32, encoder layer is set to 1, batch size $B$ is set to 128, training epoch $T$ is set to 50, number of SGD steps $M$ is set to 5, minimum cluster size $\epsilon$ is set to $\frac{N}{5*K} = 800$, weights for prior $\alpha_0$ is set to $\frac{1}{K}$, Adam with learning rate 0.005 is used for SGD optimization, we pre-train each RMTPP for 5 epochs on the whole dataset. We then set $\eta$ to $[0.1, 0.05, 0.01, 0.0064, 0.001]$, note that $0.0064 = \frac{B}{N}$. CP and ARI estimition is shown in Table 10, CP and ARI curve is shown in Fig 11.

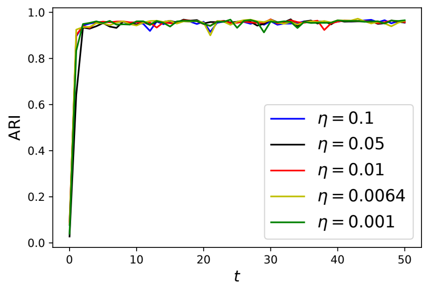
We can see that our model is not sensitive to $\eta$, a wide range of $\eta$ can give good results. For interpretability, we recommend setting $\eta = \frac{B}{N}$: $B$ sequences in $N$ cause an update of $\frac{B}{N}$.

Table 10: Performance of FATPP-MIX with respect to update rate $\eta$

| $\eta$ | 0.1 | 0.05 | 0.01 | 0.0064 | 0.001 |
|---|---|---|---|---|---|
| CP | 0.9819 | 0.9819 | 0.9828 | 0.9851 | 0.9860 |
| ARI | 0.9557 | 0.9558 | 0.9577 | 0.9633 | 0.9654 |



(a) CP



(b) ARI

Figure 11: CP and ARI curves during training process with different update rate $\eta$.