Automatic Integration for Fast and Interpretable Neural Point Processes

Anonymous Authors[1]

Abstract

The fundamental bottleneck of learning continuous-time point processes is integration. Due to the intrinsic mathematical difficulty of symbolic integration, neural point process models either constrain the intensity function to an integrable functional form or apply numerical integration. However, the former has limited expressive power, and the latter suffers additional numerical errors and high computational costs. In this paper, we introduce *Automatic Integration* for Neural point process models (AIN), a new paradigm for exact, efficient, non-parametric inference of point process. We validate our method on many synthetic temporal point process datasets and focus on the recovery of the underlying intensity function. We demonstrate that our method has clear advantages for learning irregular time series data governed by complex intensity functions. On real-world datasets with noise and unknown intensity functions, our method is also much faster than state-of-the-art neural point process models with comparable prediction accuracy.

1. Introduction

Neural point process (NPP) is a family of deep generative models that integrate deep neural networks with point processes for modeling continuous-time dynamics. NPP allows hidden states to vary between observations and is well suited for discrete event sequences such as social media posts, stock transactions, and earthquakes. The surging interests in NPP have led to the development of many state-of-the-art time series models such as Neural Hawkes process (Mei & Eisner, 2016), Neural ODE (Chen et al., 2018), see a recent survey on NPP (Shchur et al., 2021).

A central concept in point processes is the *intensity function*, which indicates the expected rates of events occurrence. Specifically, given the event sequence $\mathcal{H}_t = \{(\mathbf{s}_1, t_1), \ldots, (\mathbf{s}_n, t_n)\}_{t_n \leq t}$, the joint log-likelihood function of the observed events for point process is as follows:

$$\log p(\mathcal{H}_t) = \sum_{i=1}^{n} \log \lambda^*(t_i) - \int_0^t \lambda^*(\tau) d\tau \qquad (1)$$

where $\lambda^\star$ is the optimal intensity function.

One fundamental difficulty of maximum likelihood estimation for point processes lies in the integral term of (1). As there is no closed-form solution for the integration computation, existing approaches often use approximation. For example, Neural Hawkes process (Mei & Eisner, 2016) relies on Monte Carlo sampling, which can have high variance. Furthermore, existing NPPs often report log-likelihood as a performance measure but fail to validate the learned intensity function. We found test log-likelihood may not be a good metric not only because it is not exact, but also because learning a wrong intensity function can have little to no effect on the test likelihood.

The expressivity of NPP is another limitation. Existing methods assume that the current influence follows an expo-

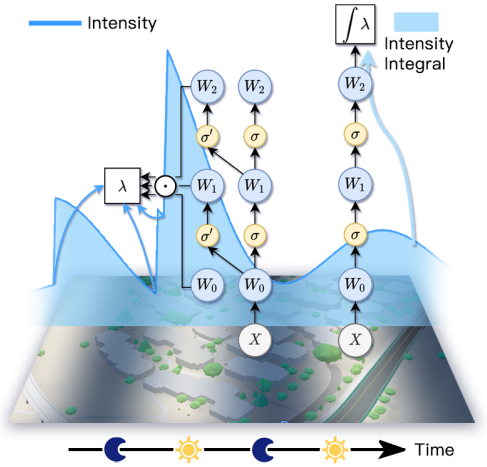

Figure 1. Illustration of learning point process with automatic integration. W denotes the linear layer's weight. σ is the nonlinear activation function. Left shows the intensity network that approximates λ and right is the integral network that computes $\int \lambda$. The two networks share the same weights.

[1]Anonymous Institution, Anonymous City, Anonymous Region, Anonymous Country. Correspondence to: Anonymous Author <anon.email@domain.com>.

Preliminary work. Under review by the International Conference on Machine Learning (ICML). Do not distribute.

nential decay of the intensity function (Du et al., 2016), or of the latent representation (Mozer et al., 2017), or even a linear interpolation (Zuo et al., 2020). Such an assumption is easily violated in real-world scenarios. An example is the delayed effect in social media posts; the influence of a viral post will not appear until several hours later, which means there could be a jump in intensity that violates the smoothness assumption. In other cases, the event influence can be cyclic, e.g., a social media bot posts every day around the same time. Both scenarios turn out to be very challenging for the existing NPP models.

To reduce the cost of integration computation and improve the expressivity of the intensity function, we ask a natural question:

> *Can we directly use a deep neural network to approximate the influence function?*

If successful, the resulting NPP would significantly relax the assumptions imposed by existing NPPs and open up new venues for modeling complex real-world event dynamics with "delayed jump" or "cyclic influence". Unfortunately, such a strategy has a major bottleneck that has prevented others from pursuing further: it requires integrating a complicated deep neural network over a large time span, where numerical integration is both inefficient and erroneous.

In this paper, we solve this problem based on the idea of automatic integration (Lindell et al., 2021; Li et al., 2019). We recognize that taking the partial derivative of a feed-forward network results in a new computational graph that shares the same set of parameters, see Figure 1. Regular NPP models use a neural network with a positive activation function to approximate the intensity. In contrast, we first construct a monotonically increasing integral network whose partial derivative is the intensity we wish to integrate. Then, we train the integral network to maximize the data likelihood. Finally, we reassemble the parameters of the integral network to obtain the intensity. This technique leads to exact solutions of the intensity and its antiderivative without imposing any constraints on their functional forms. As a result, we can efficiently compute the exact likelihood of *any* sophisticated intensity. We validate our approach using synthetic point process data with complex intensity functions, as well as a real-world earthquake dataset.

To summarize, our contributions are the following:

- We propose the first framework to speedup neural point processes learning with automatic integration. We use two networks and enforce the positivity of the intensity via a monotone integral network.
- We show that the automatic integration learns intensity functions with higher efficiency and accuracy than other NPP approaches
- We propose a simple NPP model that can recover complex influence functions, enjoys high training speed and better interpretability, and performs on par with the state-of-the-art methods on real-world data.

2. Background

Temporal Point Processes. A temporal point process (TPP) is a counting process $N(t)$, representing the number of events that occurs before time t. It is characterized by a scalar non-negative intensity function $\lambda^*(t)$. Given the history events before time t, $\mathcal{H}_t := \{t_1, ..., t_n\}_{t_n \leq t}$, the intensity function quantifies the event arrival rate at t, and is formally defined as

$$\lambda^*(t) := \lim_{\Delta t \to 0} \frac{\mathbb{E}[N(t, t+dt)|\mathcal{H}_t]}{dt}.$$

The notation $*$ is from (Daley & Vere-Jones, 2007) to indicate the intensity is conditional on the past but not including the present. One example of TPP is Hawkes process (Hawkes, 1971), characterized by

$$\lambda^*(t) = \mu + \alpha \sum_{t_i < t} \exp(-\beta(t - t_i)), \tag{2}$$

where μ, α, β are scalars. The arrival of a new event results in a sudden increase of intensity, and the influence of this event will decay exponentially. μ is the base intensity representing the rate of an event happening on its own.

Neural Point Processes. Neural Point Process (NPP) models (Mei & Eisner, 2016; Du et al., 2016; Zuo et al., 2020) combine deep neural networks with TPPs. State-of-the-art NPPs first encode the events into hidden representations using either Recurrent Neural Network (RNN) or Transformer. Then they use a non-negative activation function to map the hidden vectors to a scalar, i.e., the intensity immediately after an event. The change of intensity between events is usually represented using a linear or exponential decay function. (Mei & Eisner, 2016) allow this decay occur in a high-dimensional space before mapping to a scalar.

By parametrizing the intensity function as $\lambda^*_\theta(t)$, the log likelihood of an event sequence $\{t_1, ..., t_N\}$ observed in time interval $[0, T]$ is

$$\mathcal{L}(\theta|\{t_1, ..., t_N\}) = \sum_{i=1}^{N} \log \lambda^*_\theta(t_i^-) - \int_{t=0}^{T} \lambda^*_\theta(t).$$

As the intensity is discontinuous at every event arrival t_i, t_i^- denotes the intensity immediately before the i-th event. The second term is usually evaluated separately for each inter-event interval (t_i, t_{i+1}). If the inter-event function is

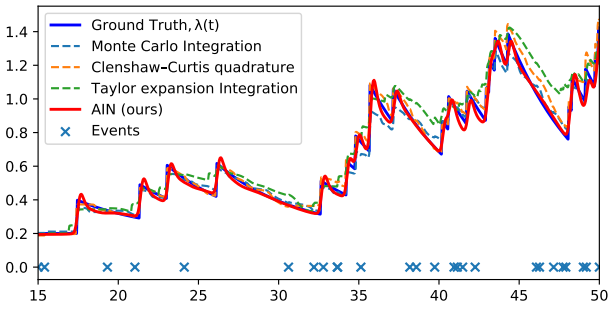

Figure 2. Intensities learned using different numerical integration methods, compared to our approach (`AIN`) with AutoInt. Blue crosses represent event over time. Numerical integration error can prevent the model from learning the truth intensity.

as simple as a scalar kernel function (Du et al., 2016), then the integral is easy, but the model is less expressive. On the other hand, if the inter-event function is high dimensional (Mei & Eisner, 2016; Zuo et al., 2020), then the model gains stronger expressive power at the cost of requiring numerical integration. We found in our experiments that the numerical integration errors may prevent the model from recovering the true underlying intensity, see Figure 2. Nevertheless, all models assume a continuous transformation of the intensity function and have limited expressivity.

3. Related Work

Parametrizing Point Process. Fitting traditional TPP models such as Hawkes process to data points may have bad performance if the model is misspecified. To address this issue, non-parametric inference for TPP has been extensively studied in the statistical literature. Early works usually rely on Bayesian methods (Møller et al., 1998; Kottas & Sansó, 2007; Cunningham et al., 2008). Rathbun & Cressie (1994) modeled the intensity function as a piecewise-constant log Gaussian. Adams et al. (2009) proposed a Markov Chain Monte Carlo (MCMC) inference scheme for the Poisson process with Gaussian priors. These Bayesian models are scalable but assume a continuous intensity change over time.

Recently, Neural Point Processes that combine TPP with neural networks has received considerable attention (Yan et al., 2018; Upadhyay et al., 2018; Huang et al., 2019; Omi et al., 2019; Shang & Sun, 2019; Zhang et al., 2020). The neural network enables the estimation of intensity after each event and significantly improves model flexibility. Under this framework, models focus more on approximating a discrete set of intensities before and after each event. The continuous intensity comes from interpolating the intensity points. For example, (Du et al., 2016) uses an RNN to generate intensities after each event. (Mei & Eisner, 2016) proposes a novel RNN architecture that generates intensities at both ends of each inter-event interval. Other works consider alternative training schema: Xiao et al. (2017) used Wasserstein distance, Guo et al. (2018) introduced noise-contrastive estimation, and Li et al. (2018) leveraged reinforcement learning. While these NPP models are more expressive than the traditional models, they still assume simple (continuous, usually monotonous) inter-event intensity changes.

Integration Methods. Integration method is largely ignored in NPP literature, but is central to a model's ability to capture the complex dynamics of a system. Existing works either used an intensity function with an elementary integral (Du et al., 2016) or used Monte Carlo integration (Mei & Eisner, 2016). However, we can see from Figure 2 that the choice of integration method has a non-trivial effect on the model performance.

Integration is generally more complicated than differentiation, which can be mechanically solved using the chain rule. Most integration rules, e.g., integration by parts and change of variables, transform an antiderivative to another that is not necessarily easier. Elementary antiderivative only exists for a small set of functions, but not even for simple composite functions such as $\exp(x^2)$ (Dunham, 2018). The Risch algorithm can determine such elementary antiderivative (Risch, 1969; 1970) but has never been fully implemented due to its complexity. The most commonly used integration methods are still numerical: Newton-Cotes Methods, Romberg Integration, Quadrature, and Monte Carlo integration (Davis & Rabinowitz, 2007).

Multiple recent works claimed for launching a new integration approach, Automatic Integration (AutoInt). Liu (2020) proposes integrating the Taylor polynomial using the derivatives from Automatic Differentiation (AutoDiff). It requires partitioning of the integral limits and choosing the order of Taylor approximation. Though it makes use of the efficient AutoDiff, the integration procedure involves a trade-off between runtime and accuracy and is numerical in nature. Li et al. (2019) and Lindell et al. (2021) proposed dual network approach which we will discuss in detail in Section 4. The method guarantees a closed-form integral and is efficient.

Monotonic Constraints. We use AutoInt for NPP, but to enforce the intensity's nonnegativity, we have to constrain its integral to be monotonically increasing. Monotonicity in neural networks has been widely studied (Archer & Wang, 1993; Doumpos & Zopounidis, 2009; Sharma & Wehrheim, 2020). There are two groups of existing approaches: network architecture design and constraints in loss.

Network architecture design usually means constraining all signs of the linear layers' weights to be positive and having a monotonic activation function. Examples include applying an element-wise exponential transformation to all linear layers' weights (Sill, 1998) or learning weights

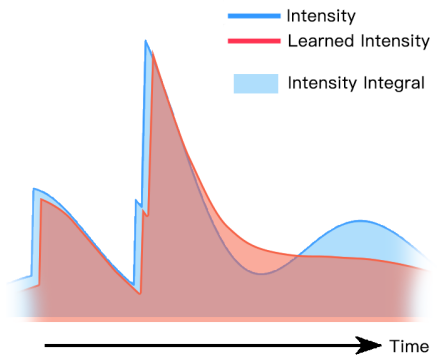

Figure 3. An example failure to restore the ground truth intensity. It is challenging because the ground truth and estimated likelihood are the same (the areas under the curves are the same.) The model needs not oversimplify the intensity to learn it correctly.

using a projected stochastic gradient descent (Chorowski & Zurada, 2014). The non-monotonic heuristics are usually approximated by randomly sampling gradients from the input domain and penalizing negative samples using hinge loss (Liu et al., 2020).

4. Methodology

In this section, we introduce a new design of the Neural Point Process, which is more interpretable and flexible. We explain how automatic integration can be used in conjunction with such an NPP for fast training and inference.

4.1. Limitations of Existing NPPs

Previous NPP models lack easy interpretation of event influence, and are not compatible with AutoInt. We begin by challenging the assumptions made by previous methods: Neural Hawkes process (Mei & Eisner, 2016) and RMTPP (Du et al., 2016).

Neural Hawkes process encodes the event history immediately after the n-th event as a hidden vector $\mathbf{h}(t_n^+) \in \mathbb{R}^k$. The model also predicts a scalar decay rate β_{t_n} and the hidden state immediately before the next event $\mathbf{h}(t_{n+1}^-)$. The model interpolates the two hidden state vectors using a kernel function f and maps the output to the intensity use a linear layer $\mathbf{w} \in \mathbb{R}^{1\times k}$. Thus, the conditional intensity function is formulated as

$$\lambda^*(t)|_{t_n\le t\le t_{n+1}} = g^+(\mathbf{w}^T f(\mathbf{h}(t_n^+), \mathbf{h}(t_{n+1}^-), \beta_{t_n})),$$

where g^+ is a positive activation function.

RMTPP also encodes the event history immediately after the n-th event as a hidden vector $\mathbf{h}(t_n^+)$, but it directly maps the vector to a scalar. The model also assumes the same intensity decay rate β applies to all events. It uses a scalar interpolation function f, and the resulting conditional intensity function is

$$\lambda^*(t)|_{t_n\le t\le t_{n+1}} = \mu + g^+(\mathbf{w}^T\mathbf{h}(t_n^+) + f(t-t_n,\beta)),$$

where μ is the scalar base intensity.

Neural Hawkes and RMTPP assume that each event's influence over the intensity is limited to the inter-event interval $[t_n^+, t_{n+1}^-]$. Their intensities are defined separately in each interval. Whereas for Hawkes process in (2), each event has a long-lasting influence; whenever a new event happens, we can decompose the summation in the intensity to analyze the contribution of any historical event to the happening of the new event, which is much more interpretable.

Both models also assume a simple change of intensity in each interval with an exponential function f. In the "cyclic influence" scenario as mentioned in Section 1, an event may have periodic reduction in influence. Either model performs poorly in this case. While the Neural Hawkes process can capture either increasing or decreasing influence, it cannot handle such a multi-modal intensity, see Figure 3. We verified that such failures are common in Section 5.

Furthermore, the designs of Neural Hawkes and RMTPP are not compatible with AutoInt. While AutoInt can approximate any function f and its antiderivative in closed forms, finding a closed-form antiderivative of $g^+ \circ f$ (where g^+ is a positive activation function) is still intractable.

4.2. Influence-Driven Point Process

We consider the following neural point process model generalizing the Hawkes process in (2):

$$\lambda^*(t) = \mu + \sum_{t_i<t} f_\theta^+(t-t_i, \mathcal{H}(t_i)). \tag{3}$$

where μ is the scalar base intensity. f_θ^+ is a positive scalar function that accepts time and $\mathcal{H}(t_i)$, i.e., a representation of the event history up to the i-th event, as inputs. The model is compatible with AutoInt as the intensity function does not contain function composition. As a result, it can efficiently evaluate definite integral over a long time span without constraining the influence to be a kernel function.

Our design has two major benefits. First, by constructing f_θ^+ as a deep neural network, our model can approximate *any* complex inter-event change of intensity, including the "cyclic influence" scenario as shown in Figure 3. Second, our model considers the long-lasting influence f of each individual event and represents the intensity in a more interpretable way. We can analyze and interpret different past events' contribution percentages to a new event by decomposing the intensity function. Whereas the previous NPP models, the past influence is a black box as it is determined arbitrarily by the hidden representation.

There are many options for representing the input $\mathcal{H}(t_i)$ to the neural network. We can directly use the difference in event times as input:

$$\lambda^*(t) = \mu + \sum_i f_\theta^+(t - t_i), f_\theta : \mathbb{R}^1 \to \mathbb{R}^1 \tag{4}$$

Alternatively, we may use a sequence model such as an RNN to encode the history into hidden vectors $\{\mathbf{h}_i\}_{i=0}^N$. The hidden vectors could be then used to scale each event's influence, such that the conditional intensity is formulated as

$$\lambda^*(t) = \mu + \sum_i g_\phi(\mathbf{h}_i) f_\theta^+(t - t_i), \tag{5}$$

$$f_\theta : \mathbb{R}^1 \to \mathbb{R}^1, \quad g_\phi : \mathbb{R}^k \to \mathbb{R}^1, \tag{6}$$

where g_ϕ is another neural network. We can also concatenate time t and $\mathbf{h}_i$ and feed them to the neural network, and the conditional intensity becomes

$$\lambda^*(t) = \mu + \sum_i f_\theta^+(t - t_i \oplus \mathbf{h}_i), \tag{7}$$

$$f_\theta : \mathbb{R}^{k+1} \to \mathbb{R}^1 \tag{8}$$

In practice, incorporating a deep sequence model greatly increases the flexibility of the model but can easily overfit. It also reduces the interpretability of the model as the influence function is in the high-dimensional domain.

4.3. Automatic Integration (AutoInt)

Suppose we have a scalar function $f_\theta(t, \mathbf{h})$ representing the intensity, we want to calculate $\int_{t=a}^b f_\theta(t, \mathbf{h}) := F_\theta(b, \mathbf{h}) - F_\theta(a, \mathbf{h})$ along one axis. AutoInt constructs the integral network F_θ first, and then reorganize the computational graph of F_θ to represent the integrant f_θ. The two networks thus share the same set of parameters θ.

Specifically, let $\mathbf{x} := t \oplus \mathbf{h}$, we consider the integral of the intensity as a fully-connected multi-layer neural network of the form

$$F_\theta(\mathbf{x}) = \mathbf{W}_n ... (\mathbf{W}_3 \sigma(\mathbf{W}_2 \sigma(\mathbf{W}_1 \mathbf{x}))),$$

where $\mathbf{W}_k : \mathbb{R}^{M_k} \mapsto \mathbb{R}^{N_k}$ denotes the weight of the k-th linear layer of the neural network and σ denotes the elementwise nonlinearity. M_k and N_k are the input and output dimension for the k-th layer. Hence, the set of parameters in this neural network is $\theta = \{\mathbf{W}_k \in \mathbb{R}^{M_k \times N_k}, \forall k\}$.

The influence network f_θ is a partial derivative of the integral network F_θ. As long as the activation function is differentiable everywhere, the intensity can be computed recursively:

$$f_\theta(\mathbf{x}) := \frac{\partial F_\theta}{\partial t}(\mathbf{x}) = \mathbf{W}_k \sigma'(\mathbf{W}_{k-1} \sigma(\mathbf{W}_{k-2} \dots (\mathbf{W}_1 \mathbf{x}))) \cdots \circ \mathbf{W}_2 \sigma'(\mathbf{W}_1 \mathbf{x}) \circ \mathbf{W}_{11}$$

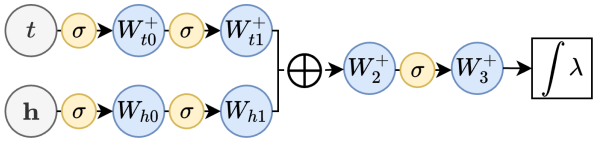

Figure 4. The architecture for monotonically increasing integral network that computes the the integral of the intensity. t is the time of the event and h is the encoded hidden vector. "W^+" indicates the neural network layer has non-negative weights.

where $\circ$ indicates the Hadamard product, and $\mathbf{W}_{11}$ is the first column of $\mathbf{W}_1$, i.e.,

$$\mathbf{W}_1 := [\mathbf{W}_{11} \quad \mathbf{W}_{12} \quad \dots \quad \mathbf{W}_{1,M_1}]$$

As noted, computing $f_\theta(\mathbf{x})$ involves many repeated operations. For example, the result of $\mathbf{W}_1 \mathbf{x}$ is used for compute both $\sigma(\mathbf{W}_1 \mathbf{x})$ and $\sigma'(\mathbf{W}_1 \mathbf{x})$, see Figure 1. Therefore, we implemented a program that leverages dynamical programming to efficiently create a derivative model using automatic differentiation. During training, we use the two networks to calculate the likelihood of event sequences, as if they are regular neural networks.

4.4. Imposing the Non-negativity Constraint

We constrain the intensity to be non-negative by forcing its integral network to be monotonically increasing. As this is a strict constraint, we cannot use regularization by penalizing negative gradients; otherwise, the negative intensity would lead to erroneous log-likelihood. Also, we cannot simply constrain all linear weights in the network to be non-negative because we want the network to be monotonic only for the time input but not others.

We design the following architecture for the integral network, as illustrated in Figure 4: we first pass the hidden vector $\mathbf{h}$ and the time t through two linear layers with non-negative weights W^+ separately. Then, we concatenate the outputs to another non-negative weighted network. Therefore, the resulting integral monotonically increases with respect to time, as the time input t does not pass through any layer with negative weights. The two unconstrained layers with weights W also ensure the expressivity of other input dimensions is not impaired.

We experimented with different ways to enforce positive weights. We found that projected gradient descent (i.e., clamping the weights after each optimizer step) converges to the ground truth better than the exponential transformation method. To ensure monotonicity, we need to use monotonic activation function; previous AutoInt works use sine activation (Lindell et al., 2021) which is non-monotonic. We found that tanh and sine activations yield similar performance, as also indicated by Parascandolo et al. (2016).

Model	shakyHawkes		shiftHawkes		decayPeak		earthquakesJP
	MAPE	LL	MAPE	LL	MAPE	LL	LL
CT-GRU (Mozer et al., 2017)	0.2243	-35.6063	0.1262	-39.7173	0.1103	-42.1959	5.9148
Neural Hawkes (Mei & Eisner, 2016)	0.2168	-35.4043	0.1473	-40.0411	0.1468	-42.5548	7.0620
RMTPP (Du et al., 2016)	0.2562	-35.6549	0.2630	-39.7893	0.2183	-42.7965	7.7663
Transformer Hawkes (Zuo et al., 2020)	0.2812	-36.1831	0.2316	-40.6717	0.2342	-43.3308	6.0588
AIN	**0.1843**	**-35.3762**	**0.0356**	**-39.3599**	**0.0226**	**-41.9678**	**8.6187**
AIN (w/ RNN)	0.3353	-37.9182	0.4675	-44.0076	0.1107	-42.3124	7.7730

Table 1. Comparison between our proposed model AIN (with or without RNN) and the state-of-the-art NPP models on three synthetic datasets and a real-world *Earthquake Japan* dataset. Performance w.r.t. Mean Absolute Percentange Error (MAPE) of the estimated conditional intensity $\lambda^*(t)$ and Test log likelihood (LL).

4.5. Loss Function

Given the monotonic integral network $F_\theta(t, \mathbf{h})$ and the event influence network $f_\theta = \frac{\partial F_\theta}{\partial t}$ obtained from AutoInt, the log-likelihood of an event sequence $\{t_1, ..., t_N\}$ observed in time interval $[0, T]$ with respect to the model is

$$\begin{aligned}&\mathcal{L}(\{t_1, ..., t_N\}, \{\mathbf{h}_0, ..., \mathbf{h}_N\})\\ &= \sum_{i=1}^{N} \log\left(\sum_{j=1}^{i-1} f_\theta(t_i - t_j, \mathbf{h}_i)\right) + \\ &\sum_{i=1}^{N} [F_\theta(T - t_i, \mathbf{h}_N) - F_\theta(0, \mathbf{h}_i)],\end{aligned}$$

where $\{\mathbf{h}_0, ..., \mathbf{h}_N\}$ are the latent representations generated by a deep sequence model. This is straightforward by the Fundamental Theorem of Calculus. In case where the deep sequence model is not used, the log-likelihood evaluates to

$$\begin{aligned}\mathcal{L}(\{t_1, ..., t_N\}) = &\sum_{i=1}^{N} \log\left(\sum_{j=1}^{i-1} f_\theta(t_i - t_j)\right) + \\ &\sum_{i=1}^{N} [F_\theta(T - t_i) - F_\theta(0)].\end{aligned}$$

We can learn the parameters θ in both networks by maximizing the log-likelihood function.

5. Experiments

We evaluate AIN for learning temporal dynamics using both synthetic and real-world data.

5.1. Experimental Setup

Synthetic Datasets. Existing temporal point process models fail to capture the dynamics of the point process governed by multimodal or non-smooth current influence function. Here, we proposed and simulated three challenging synthetic datasets using Ogata's thinning algorithm (Chen, 2016).

- *Shaky Hawkes process*: as illustrated by Figure 3, it multiplies the influence function of the Hawkes process (see Equation 2) by a cyclic function, such that the intensity becomes multimodal in long inter-event intervals. It is characterized by the conditional intensity function

  $$\lambda^*(t) = \mu + \alpha \sum_{i=1}^{N} \cos((t - t_i) + 1) \exp(-\beta(t - t_i))$$

  In our experiments, we set $\alpha = \beta = \mu = 0.2$.

- *Delayed Peak process*: it features a uni-modal but non-smooth influence function. Each event's influence is initially 0; then it first increases and then decreases, following a bell-shaped curve. It is characterized by

  $$\lambda^*(t) = \mu + \alpha \sum_{i=1}^{N} \text{ReLU}(-(\beta(t - t_i) - 1)^2 + 1)$$

  In our experiments, we set $\alpha = 0.2, \beta = 0.5, \mu = 0.3$.

- *Shift Hawkes process*: as mentioned in Section 1, it describes the scenario in which a post becomes viral several hours after it is visible, such that there is a jump in the intensity between events. It is characterized by

  $$\lambda^*(t) = \mu + \alpha \sum_{i=1}^{N} \mathbf{1}(t - t_i > \gamma) \exp(-\beta(t - t_i - \gamma))$$

  In our experiments, we set $\alpha = \beta = \mu = 0.2$ and the threshold $\gamma = 2.0$.

Each synthetic dataset contains 8192 sequences over a time range of $[0, 50)$. The train-val-test split is $2 : 1 : 1$.

Real-world Datasets. We use a real-world dataset, *Earthquake Japan*, that includes the times and locations of all

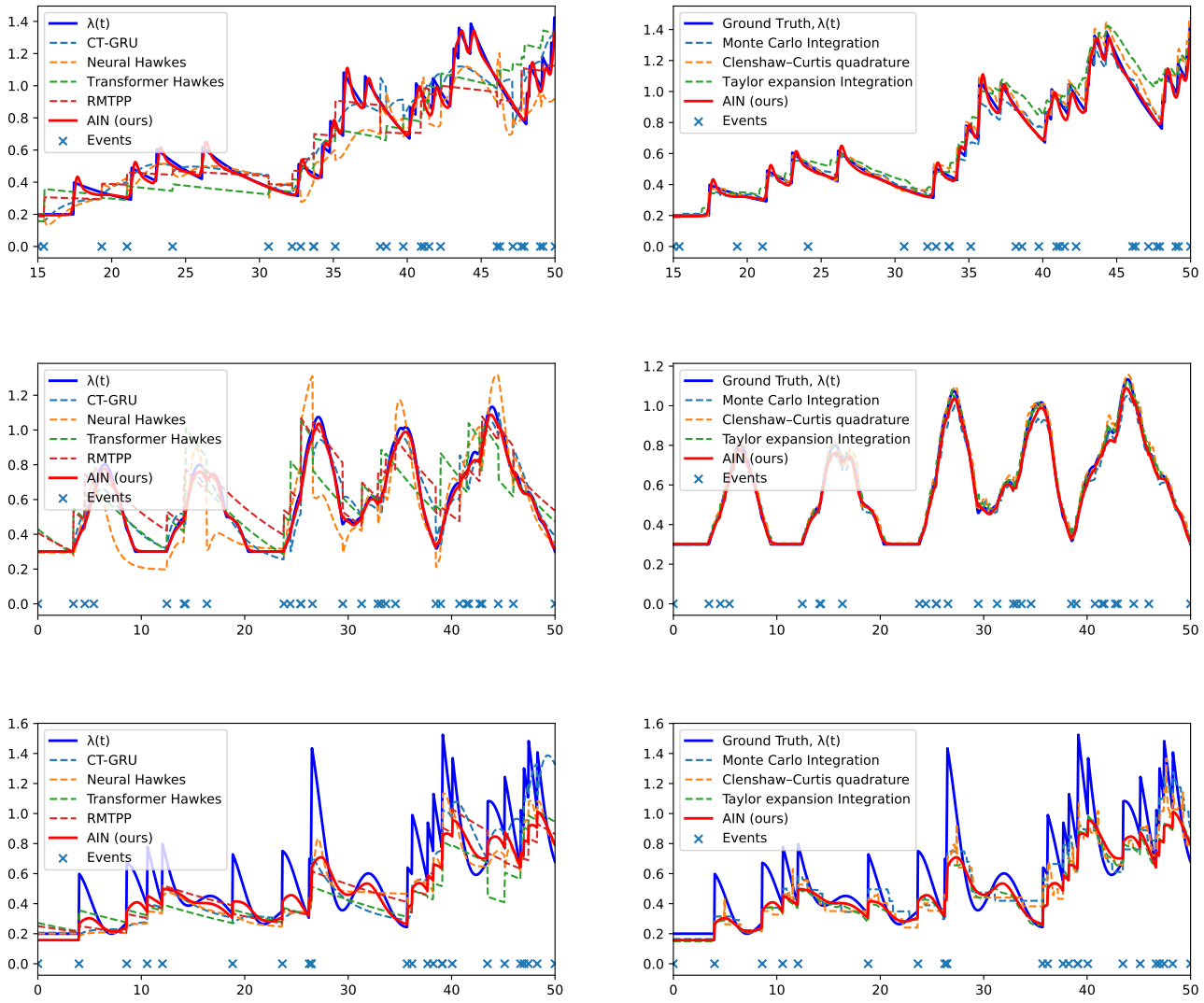

Figure 5. Visualizations of the true conditional intensity $\lambda^*(t)$ and the learned conditional intensity on the *Shift Hawkes* (first row), *Delayed Peak* (second row), *Shaky Hawkes* (third row) datasets. First column: comparison of intensities learned with different models. Second column: comparison of intensities learned with different integration methods.

Model	shakyHawkes		shiftHawkes		decayPeak		earthquakesJP
	MAPE	LL	MAPE	LL	MAPE	LL	LL
Clenshaw-Curtis	0.2197	-35.5183	0.0541	-39.4831	0.0312	-41.9839	8.4299
Monte Carlo	0.1935	-35.6090	0.0462	**-39.3527**	0.0378	-41.9868	8.1906
Taylor Expansion (Liu, 2020)	0.2004	-35.3771	0.0999	-39.7062	**0.0224**	-41.9691	7.7142
AIN	**0.1843**	**-35.3762**	**0.0356**	-39.3599	0.0226	**-41.9678**	**8.6187**

Table 2. Comparison between different integration methods on three synthetic datasets and the *Earthquake Japan* dataset. Performance w.r.t. Mean Absolute Percentange Error (MAPE) of the estimated conditional intensity $\lambda^*(t)$ and Test log likelihood (LL).

earthquakes in Japan from 1990 to 2020 with magnitudes of at least 2.5. It is gathered by Chen et al. (2020). The dataset contains 1500 sequences over a time range of $[0, 30)$. The train-val-test split is $4:1:1$.

Evaluation Metrics. As shown by Figure 3, a TPP model may yield a likelihood similar to the ground truth but fail to capture the correct intensity. Therefore, in addition to the likelihood (LL), we show the Mean Absolute Percentage Error (MAPE) of the estimated conditional intensity.

Baselines. We have two groups of baselines:

- *Integration methods*: the baselines learn the point process using the same model as described by Equation 4 but with four different integration techniques: Taylor integration ((Liu, 2020), see Appendix A.1), the Clenshaw–Curtis quadrature (see Appendix A.2), the Monte Carlo integration, and AutoInt.

- *State-of-the-art approaches*: they are state-of-the-art NPP models, including RMTPP (Du et al., 2016), Neural-Hawkes (Mei & Eisner, 2016) and Transformer Hawkes (Zuo et al., 2020). Additionally, (Mozer et al., 2017) proposed a continuous-time GRU that interpolates hidden states between events. It has a similar idea as Neural-Hawkes's continuous-time LSTM. We include a CT-GRU variant of Neural-Hawkes to increase the diversity of our baselines.

5.2. Experimental Results

Figure 6 visualizes the run-time comparison of different methods. We can see that AutoInt trains faster than other numerical integration techniques, and AIN is much faster than most state-of-the-art models.

Table 1 compares the prediction Mean Absolute Percentage Error (MAPE) and test log-likelihood (LL) between AIN and the state-of-the-art models on the four datasets. Table 2 compares the evaluation metrics between AutoInt and other numerical integration methods using the same model. We can see that AIN has a decisive advantage on the synthetic dataset with complex intensity. We can also tell from the bottom-left of subfigure of Figure 5 that our method is the only one that can capture the multimodal intensity function.

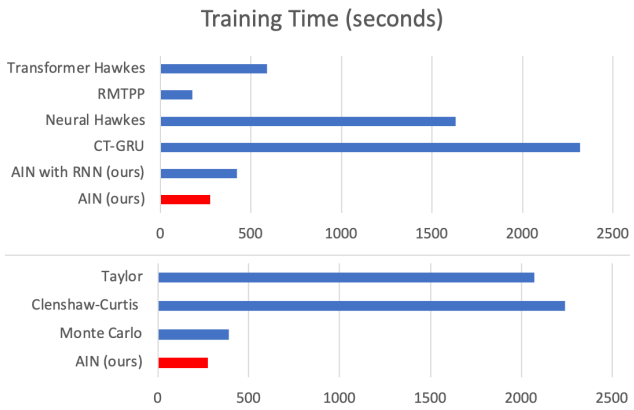

Figure 6. Training speed comparison for different NPPs and numerical integration methods in seconds. The proposed AIN is fast. RMTPP is the fastest but suffers from poor prediction performance.

6. Conclusion

We propose Automatic Integration for Neural point process models (AIN) using a dual network approach. AIN can efficiently compute the exact likelihood of *any* sophisticated intensity. We validate our approach using many synthetic data with complex intensity functions, and a real-world earthquake dataset. Experiment results demonstrate that AIN can efficiently and accurately recover the underlying intensity function.

Our work presents a new paradigm for learning continuous-time dynamics. Currently our neural process model takes the form of Hawkes processes (self-exciting) but cannot handle self-correcting processes due to the difficulty of integration. Future work includes relaxing the form of intensity network with advanced integration technique. Another interesting direction is to generalize the 1-D temporal intensity function to 3-D spatiotemporal intensity function to model spatiotemporal point processes.

References

Adams, R. P., Murray, I., and MacKay, D. J. Tractable nonparametric bayesian inference in poisson processes with gaussian process intensities. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pp. 9–16, 2009.

Archer, N. P. and Wang, S. Application of the back propagation neural network algorithm with monotonicity constraints for two-group classification problems. *Decision Sciences*, 24(1):60–75, 1993.

Chen, R. T., Rubanova, Y., Bettencourt, J., and Duvenaud, D. Neural ordinary differential equations. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, pp. 6572–6583, 2018.

Chen, R. T., Amos, B., and Nickel, M. Neural spatio-temporal point processes. *arXiv preprint arXiv:2011.04583*, 2020.

Chen, Y. Thinning algorithms for simulating point processes. *Florida State University, Tallahassee, FL*, 2016.

Chorowski, J. and Zurada, J. M. Learning understandable neural networks with nonnegative weight constraints. *IEEE transactions on neural networks and learning systems*, 26(1):62–69, 2014.

Cunningham, J. P., Shenoy, K. V., and Sahani, M. Fast gaussian process methods for point process intensity estimation. In *Proceedings of the 25th international conference on Machine learning*, pp. 192–199, 2008.

Daley, D. J. and Vere-Jones, D. *An introduction to the theory of point processes: volume II: general theory and structure*. Springer Science & Business Media, 2007.

Davis, P. J. and Rabinowitz, P. *Methods of numerical integration*. Courier Corporation, 2007.

Doumpos, M. and Zopounidis, C. Monotonic support vector machines for credit risk rating. *New Mathematics and Natural Computation*, 5(03):557–570, 2009.

Du, N., Dai, H., Trivedi, R., Upadhyay, U., Gomez-Rodriguez, M., and Song, L. Recurrent marked temporal point processes: Embedding event history to vector. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pp. 1555–1564, 2016.

Dunham, W. *The calculus gallery*. Princeton University Press, 2018.

Guo, R., Li, J., and Liu, H. Initiator: Noise-contrastive estimation for marked temporal point process. In *IJCAI*, pp. 2191–2197, 2018.

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

Huang, H., Wang, H., and Mak, B. Recurrent poisson process unit for speech recognition. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pp. 6538–6545, 2019.

Kottas, A. and Sansó, B. Bayesian mixture modeling for spatial poisson process intensities, with applications to extreme value analysis. *Journal of Statistical Planning and Inference*, 137(10):3151–3163, 2007.

Li, H., Li, Y., and Li, S. Dual neural network method for solving multiple definite integrals. *Neural computation*, 31(1):208–232, 2019.

Li, S., Xiao, S., Zhu, S., Du, N., Xie, Y., and Song, L. Learning temporal point processes via reinforcement learning. *arXiv preprint arXiv:1811.05016*, 2018.

Lindell, D. B., Martel, J. N., and Wetzstein, G. Autoint: Automatic integration for fast neural volume rendering. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 14556–14565, 2021.

Liu, K. Automatic integration. *arXiv e-prints*, pp. arXiv–2006, 2020.

Liu, X., Han, X., Zhang, N., and Liu, Q. Certified monotonic neural networks. *arXiv preprint arXiv:2011.10219*, 2020.

Mei, H. and Eisner, J. The neural hawkes process: A neurally self-modulating multivariate point process. *arXiv preprint arXiv:1612.09328*, 2016.

Møller, J., Syversveen, A. R., and Waagepetersen, R. P. Log gaussian cox processes. *Scandinavian journal of statistics*, 25(3):451–482, 1998.

Mozer, M. C., Kazakov, D., and Lindsey, R. V. Discrete event, continuous time rnns. *arXiv preprint arXiv:1710.04110*, 2017.

Omi, T., Ueda, N., and Aihara, K. Fully neural network based model for general temporal point processes. *arXiv preprint arXiv:1905.09690*, 2019.

Parascandolo, G., Huttunen, H., and Virtanen, T. Taming the waves: sine as activation function in deep neural networks. 2016.

Rathbun, S. L. and Cressie, N. Asymptotic properties of estimators for the parameters of spatial inhomogeneous poisson point processes. *Advances in Applied Probability*, 26(1):122–154, 1994.

Risch, R. H. The problem of integration in finite terms. *Transactions of the American Mathematical Society*, 139: 167–189, 1969.

Risch, R. H. The solution of the problem of integration in finite terms. *Bulletin of the American Mathematical Society*, 76(3):605–608, 1970.

Shang, J. and Sun, M. Geometric hawkes processes with graph convolutional recurrent neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pp. 4878–4885, 2019.

Sharma, A. and Wehrheim, H. Testing monotonicity of machine learning models. *arXiv preprint arXiv:2002.12278*, 2020.

Shchur, O., Türkmen, A. C., Januschowski, T., and Günnemann, S. Neural temporal point processes: A review. *arXiv preprint arXiv:2104.03528*, 2021.

Sill, J. Monotonic networks. 1998.

Upadhyay, U., De, A., and Gomez-Rodriguez, M. Deep reinforcement learning of marked temporal point processes. *arXiv preprint arXiv:1805.09360*, 2018.

Xiao, S., Farajtabar, M., Ye, X., Yan, J., Song, L., and Zha, H. Wasserstein learning of deep generative point process models. *arXiv preprint arXiv:1705.08051*, 2017.

Yan, J., Liu, X., Shi, L., Li, C., and Zha, H. Improving maximum likelihood estimation of temporal point process via discriminative and adversarial learning. In *IJCAI*, pp. 2948–2954, 2018.

Zhang, Q., Lipani, A., Kirnap, O., and Yilmaz, E. Self-attentive hawkes process. In *International Conference on Machine Learning*, pp. 11183–11193. PMLR, 2020.

Zuo, S., Jiang, H., Li, Z., Zhao, T., and Zha, H. Transformer hawkes process. In *International Conference on Machine Learning*, pp. 11692–11702. PMLR, 2020.

A. Appendix

A.1. Taylor Integration

In this section, we briefly introduce the algorithm proposed by Liu (2020) and our implementation of it. Let $\beta_1, \ldots, \beta_5$ be real scalar hyperparameters with $\beta_1 \neq 0$. They can be viewed as the coefficients of the Taylor remainders. In practice, we set $\beta_1 = 1$ and $\beta_2 = \beta_3 = \beta_4 = \beta_5 = 0$. We tried different sets of β, but we found that their influence is trivial in most case. If the real numbers $A_1, ..., A_5$ are given by

$$\begin{cases} A_5 := \frac{(b-c)^6-(a-c)^6}{6\beta_1^5} \\ A_4 := \frac{(b-c)^5-(a-c)^5}{5\beta_1^4} - \frac{4\beta_2 A_5}{\beta_1}, \\ A_3 := \frac{(b-c)^4-(a-c)^4}{4\beta_1^3} - \frac{3\beta_2 A_4}{\beta_1} - 3\left(\frac{\beta_3}{\beta_1} + \frac{\beta_2^2}{\beta_1^2}\right) A_5, \\ A_2 := \frac{(b-c)^3-(a-c)^3}{3\beta_1^2} - \frac{2\beta_2 A_3}{\beta_1} - \left(\frac{2\beta_3}{\beta_1} + \frac{\beta_2^2}{\beta_1^2}\right) A_4 + \\ -2\left(\frac{\beta_4}{\beta_1} + \frac{\beta_2\beta_3}{\beta_1^2}\right) A_5, \\ A_1 := \frac{(b-c)^2-(a-c)^2}{2\beta_1} - \frac{\beta_2 A_2}{\beta_1} - \frac{\beta_3 A_3}{\beta_1} - \frac{\beta_4 A_4}{\beta_1} - \frac{\beta_5 A_5}{\beta_1}, \end{cases}$$

and $y_1(c), ..., y_5(c)$ are given by evaluation of function and its derivatives

$$\begin{aligned} &\underbrace{f(c)}_{y_0} + \underbrace{\beta_1 f'(c)}_{y_1}\varepsilon + \underbrace{\left\{\beta_2 f'(c) + \frac{1}{2!}\beta_1^2 f^{(2)}(c)\right\}}_{y_2}\varepsilon^2 + \underbrace{\left\{\beta_3 f'(c) + \beta_1\beta_2 f^{(2)}(c)\right.}_{y_3} + \underbrace{\left.\frac{1}{3!}\beta_1^3 f^{(3)}(c)\right\}}_{y_3}\varepsilon^3 \\ &+ \underbrace{\left\{\beta_4 f'(c) + \left(\beta_1\beta_3 + \frac{1}{2}\beta_2^2\right) f^{(2)}(c) + \frac{1}{2}\beta_1^2\beta_2 f^{(3)}(c)\ \ \frac{1}{4!}\beta_1^4 f^{(4)}(c)\right\}}_{y_4}\varepsilon^4 \\ &+ \underbrace{\left\{\beta_5 f'(c) + (\beta_1\beta_4 + \beta_2\beta_3) f^{(2)}(c)\frac{1}{2}\left(\beta_1^2\beta_3 + \beta_1\beta_2^2\right) f^{(3)}(c) + \frac{1}{6}\beta_1^3\beta_2 f^{(4)}(c) + \frac{1}{5!}\beta_1^5 f^{(5)}(c)\right\}}_{y_5}\varepsilon^5 \end{aligned}$$

The function $\int_a^b f(x)$ can be approximated by $\sum_k \int_a^b y_k(x)A_k$. We calculated the n-th derivative using PyTorch AutoDiff. When using the Taylor Integration for learning point process, we equally split the time interval to subintervals of width 0.1 (which means at most 500 subintervals on the synthetic dataset) and evaluate the integral over each subinterval.

A.2. Clenshaw-Curtis Quadrature

In this section, we briefly introduce the Clenshaw-Curtis algorithm, which integrates a function $f(x)$ by first approximating it using a linear combination of Chebyshev polynomials $T_n(x)$, which follows the recurrence relationship

$$T_0(x) = 1, T_1(x) = x, T_{n+1}(x) = 2xT_n(x) - T_{n-1}(x)$$

or generally,

$$T_n(x) = \cos\left(n\cos^{-1}x\right)$$

The roots of the Chebyshev polynomials are Chebyshev nodes, where

$$x_k = \cos\left(\frac{2k-1}{2n}\pi\right), \quad k = 1, \ldots, n$$

We can use affine transformation to map the integrant from $x \in [a, b]$ to $x \in [-1, 1]$ and evaluate $(b-a)\sum\int_{-1}^{1} T_n(x)$ as an estimator of $\int_a^b f(x)$. Consider the average of function evaluated at nodes, $y = f(\bar{x}_k)$. The Chebyshev coefficients are

$$c_0 = \frac{1}{K}\sum f(x_k), c_n|_{n\neq 0} = \frac{2}{K}\sum T_n(x_k)f(x_k),$$

where x_k are the Chebyshev nodes. The integral of a Chebyshev polynomial is

$$\begin{aligned}
\int T_n(x)dx &= \int T_n(\cos\theta)d\cos\theta \\
&= -\int \cos(n\theta)\sin\theta d\theta \\
&= -\frac{1}{2}\int (\sin((n+1)\theta) - \sin((n-1)\theta))d\theta \\
&= \frac{1}{2}\left(\frac{\cos((n+1)\theta)}{n+1} - \frac{\cos((n-1)\theta)}{n-1}\right) + \text{const.} \\
&= \frac{1}{2}\left(\frac{T_{n+1}(x)}{n+1} - \frac{T_{n-1}(x)}{n-1}\right) + \text{const.}
\end{aligned}$$

Finally, we sum of the integral of each Chebyshev polynomial to obtain an estimate of $\int_a^b f(x)$. In our implementation, we use a default number of 1000 Chebyshev nodes when integrating the influence function.