AUTOMATIC INTEGRATION FOR NEURAL POINT PROCESSES

Anonymous authors
Paper under double-blind review

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 certain numerical methods. 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, 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 temporal 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

Social media posts, stock transactions, COVID infections — millions of event sequences are being generated every day. Unlike regular time series, event sequences are irregularly sampled with missing values. Many deep sequence models such as RNN, LSTM (Hochreiter & Schmidhuber, 1997) or Transformer (Vaswani et al., 2017) do not emphasize on the time intervals between events, only updating the hidden representation whenever events occur. In contrast, neural point process (NPP) models including Neural Hawkes Mei & Eisner (2016), Neural ODE Chen et al. (2018) allow hidden representation to vary between events and are better suited for modeling event sequences.

Neural point processes combine deep sequence models with point processes. However, existing NPPs often fail to validate that the model can indeed *recover the true intensity function*. Even for those with synthetic experiments (Du et al., 2016; Shchur et al., 2019), the validation is only limited to monotone intensity changes, such as Hawkes and self-correcting processes. Hence, it remains an open question whether NPP can accurately pick up the subtlety in the inter-event intensity change and infer the true influence function. Furthermore, log-likelihood is a gold standard for evaluating NPPs, but we found test log-likelihood may not be a good metric. Because learning a wrong intensity function can have little to no effect on the test likelihood, as we will show in Section 4.

The expressivity of NPP is another limitation. Existing methods assume that the current influence follows an exponential 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. Consider the delayed jump effect in social media posts, the content of a viral post will not be visible until several hours later. 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 very challenging for the existing NPP models.

As deep neural networks are universal functional approximators. The question is: *can we directly use deep neural networks to approximate the influence functions in point processes?* If successful, the resulting NPP would significantly relax the assumption imposed by existing NPPs and open up new venues to model complex real-world event dynamics with "delayed jump" or "cyclic influence". Unfortunately, such a strategy has a major bottleneck that has prevented others from pursing further: it requires the integration of a complicated deep neural network over a large time span, where numerical integration is both inefficient and prone to numerical errors.

In this paper, we introduce the automatic integration technique (Lindell et al., 2021; Li et al., 2019) to the field of point process to effectively solve this problem. Our method takes a dual network approach, a gradient network and an integral network. The gradient network is trained... . We validate on three different temporal point processes using the automatic integration technique and notice that some constraints (e.g. same influences for events) still need to be enforced in order to let the model learn correctly. Nonetheless, with the constraint, the model is able to capture complicated dynamics with high time efficiency.

To summarize, our contributions are the following:

- We first combine neural temporal point processes and automatic integration, introducing ways to enforce intensity function's positivity given that the original technique does not allow controlling integrant's functional form.
- We show the automatic integration efficiently learns smoother intensity function with higher accuracy, compared to traditional numerical methods.
- We propose a simple RNN-free model that can recover relatively complicated influence functions, enjoys high training speed 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 occurred before time t. It is characterized by a scalar non-negative intensity function $\lambda^*(t)$. Given a infinitesimal half-open time window $[t, t+dt)$, conditioning on the history events before time t, $\mathcal{H}_t := \{t_1, ..., t_n\}_{t_n \leq t}$, the intensity function means 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. A example of TPP is Hawkes process, characterized by $\lambda^*(t) = \mu + \alpha \sum_{t_i < t} \exp(-\beta(t - t_i))$, where β is a scalar parameter. Arrival of a new event results in a sudden increase of α, and the influence of this event will decay exponentially with the rate β over time. μ is the base intensity meaning the rate of 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 sequential models with TPP. State-of-the-art NPPs first encode the events into hidden representations using either RNN or Transformer, then model the conditional intensity function as

$$\lambda^*(t) = \mu + f^+(\mathbf{w}^T \mathbf{h} + f_{\text{current}}(t, t_n)), \quad \lambda^*(t) = f^+(\mathbf{w}^T f_{\text{current}}(\mathbf{h}, t, t_n))$$

Here t_n is the time of the last event, $\mathbf{w}^T$ maps the hidden representation to a scalar intensity, f^+ ensures the intensity is positive. The function f_{current} represents the current influence of the intensity or the hidden states from t to t_n. f_{current} is usually assumed to be an exponential decay function, which means the intensity update between event is monotone or uni-modal.

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

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

Due to the presence of integration, designing the appropriate form of the intensity function $\lambda^*_\theta(t)$ becomes essential. If the intensity function is simple such as an exponential decay (Du et al., 2016), then the integration has a closed form. But this also limits the expressive power of the NPP model. If the intensity function is complex, then numerical integration is required (Mei & Eisner, 2016; Zuo et al., 2020). We found in our experiments that the numerical errors may prevent the model from recovering the true underlying intensity function.

Automatic Integration Integration is fundamentally more difficult than differentiation. Unlike differentiation which can always break down composite functions using the chain rule, integration has no effective way to decompose composite functions. Closed-form antiderivative solutions only exist for a small set of functions, and when they exist, different algorithms other than back-propagation are required for finding them, such as the Risch-Norman algorithm (Risch, 1969). This makes the direct integration of a deep neural network intractable.

Using deep neural networks for automatic integration have been investigated in (Teichert et al., 2019; Li et al., 2019; Lindell et al., 2021). A common idea is think in reverse: transforming integration to learning the weights for the gradient of another neural network. Specifically, an integral network is first constructed, and then a gradient network is formed by reassembling the integral network's computational graph. The gradient network is the partial derivative of the integral network w.r.t. a specific input dimension. Thus, the definite integral can be computed by the fundamental theorem of calculus, using the integral network's values at the endpoints. A drawback of this dual network approach is that we cannot control the form of the integrant, e.g. add an exponential layer to it to make it positive. In other words, after we know the antiderivative of a function, it is clear that finding the antiderivative of the exponential of the function is still intractable.

Numerical integration, including Riemann integral, Monte Carlo sampling, and orthogonal polynomials (Davis & Rabinowitz, 2007) are not "automatic". Liu (2020) propose Taylor approximation using derivatives from auto-differentiation and also name it automatic integration; it still requires partitioning of the integral interval. Such a method requires partitioning of the integral interval, multiple computational graphs and is computationally expensive.

3 METHODOLOGY

3.1 MODEL DEFINITION

We consider the following neural point process model:

$$\lambda^*(t) = \mu + \sum_i f^+_{\text{NN}}(t - t_i, \mathcal{H}_{t_i}) \tag{1}$$

Where μ is a scalar parameter representing the base intensity. We model the event intensity as the sum of the influence from all the past events, each represented by a neural network function f^+_{NN} with a positive scalar output. $\mathcal{H}_{t_i}$ is the entire history before event i.

Note that the NPP in Equation 1 directly models the influence of individual event separately which is infeasible for existing NPPs. Because for simple influence functions, it usually involves a monotone decay over time, far-away events would have diminishing influence. For complex influence function where numerical integration is required, a large amount of samples is required to evaluate the very first event's influence on the final event. Most models opt for using deep sequence models to embed the history and only consider latest event. In contrast, thanks to automatic integration, our model can efficiently evaluate definite intervals over a long time span without such a constraint.

Our design has two major benefits. First, as neural networks are universal function approximators, our model is able to approximate any inter-event change in intensity when a new event arrives, including the aforementioned "delayed jump" and "cyclic influence" scenarios. Second, the model is more interpretable. We can analyze different past events' contribution percentages to a new event because the intensity function is fully decomposable. Whereas for those with deep sequence models, the past influence is a black-box as it is determined arbitrarily by the hidden representation.

There are many options for representing $\mathcal{H}_{t_i}$ as an input to the neural network. The first one is to simply ignore $\mathcal{H}_{t_i}$ and to assume all events share the same feed-forward neural network function, hence the model *AutoInt Process with same influence*:

$$\lambda^*(t) = \mu + \sum_i f^+_{NN}(t - t_i), \quad f_{NN} : \mathbb{R}^1 \to \mathbb{R}^1 \tag{2}$$

Such a model is quite limited in its expressivity. Because there are no hidden states in the f_{NN}, other contextual information of the event can not be easily integrated into the model.

An alternative is to still use an RNN or Transformer to encode the history $\mathcal{H}_{t_i}$ into hidden representation $\mathbf{h}_i$, hence the model *AutoInt Process with unconstrained influence*:

$$\lambda^*(t) = \mu + \sum_i f^+_{NN}(t - t_i \oplus \mathbf{h}_i), \quad f_{NN} : \mathbb{R}^{\text{hidden}+1} \to \mathbb{R}^1 \tag{3}$$

This design is highly expressive. However, it may also require strong regularization, since the model could easily overfit. Recall that the TPP likelihood is $\sum \log \lambda^*(t_i) - \int_0^T \lambda^*(t)$. The model in Equation 3 could drive the likelihood to infinity by having extremely high intensity when an event is going to happen and 0 intensity elsewhere.

The third option is a balanced one: we assume all events to share the same influence function, meanwhile, allow the magnitude of influence to vary based on the hidden state, hence the model *AutoInt-RNN Process with same influence*:

$$\lambda^*(t) = \mu + \sum_i g_{NN}(\mathbf{h}_i) f^+_{NN}(t - t_i), \quad f_{NN} : \mathbb{R}^1 \to \mathbb{R}^1, \quad g_{NN} : \mathbb{R}^{\text{hidden}} \to \mathbb{R}^1 \tag{4}$$

This setting provides additional regularization compared to the former one .

3.2 Reassembling of the Gradient networks

The composition of the gradient network that shares the parameter with the integral network can be implemented by recursion according to the chain rule.

Here we give a brief example of how AutoInt's reassembling works. Consider

$$F_{NN}(x) = W_2 \sin(W_1 x),$$

the gradient network is

$$f_{NN}(x) := F'_{NN}(x) = W_2 \cos(W_1 x) \cdot W_1$$

3.3 Imposing the Non-negativity Constraint

The core idea of automatic integration is to construct the integral network first, and then reassemble the computational graph to represent the integrant. Since the integrant network comes from the reassembling, one cannot add a final layer to it. Nonetheless, state-of-the-art models usually enforce intensity's non-negativity by having a final exponential or softplus layer (Du et al., 2016; Mei & Eisner, 2016; Zuo et al., 2020).

However, we have full control over the construction of the integral network. The intensity function would be non-negative if we constrain the integral network to be monotonically increasing. The monotonic neural network has been a well-researched topic for improving the model's robustness to adversarial attacks. There are two groups of approaches for enforcing network monotonicity (Liu et al., 2020): 1. guarantee monotonicity by construction and 2. use certain heuristics to detect non-monotonic part of the network (e.g. negative gradient) and include in the loss function.

Since when there are negative intensities, the temporal point process model is ill-defined, we have to guarantee monotonicity by construction, which is usually done by constraining all signs of the linear layers' weights to be positive and having a monotonic activation function. The weight sign constraints are usually done by either using an elementwise exponential transformation (Sill, 1998) or a projected stochastic gradient descent (setting all weights to positive after each step) (Chorowski & Zurada, 2014). The monotonicity of the activation function means sine function normally used as AutoInt's activation function is no longer applicable (Lindell et al., 2021). Since automatic integration involves the gradient network, we would like the activation function to be infinitely differentiable; sine function is thus a good choice in normal settings.

We empirically determined that for the task of fitting a non-negative integrant network with a function and obtain the function's definite integral, the projected stochastic gradient descent converges to the ground truth stabler than the exponential transformation method. Also, tanh activation has similar performance with sine activation, as also indicated by Parascandolo et al. (2016).

3.4 LOSS FUNCTION

Given the constructed monotonic intensity integral network F_{NN}, the influence function is $f_{NN} = \frac{\partial f_{NN}}{\partial t}$ obtained from reassembling the computational graph of F_{NN}. The log-likelihood of an event sequence $\{t_1, ..., t_N\}$ observed in time interval $[0, T]$ with respect to the model is

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

4 EXPERIMENTS

Through AutoInt can be directly implemented by Pytorch, it would suffer additional performance penalties (Lindell et al., 2021), so we implement our version of AutoInt inheriting the Pytorch's sequential network.

4.1 DATASETS

In the Introduction section, we mentioned the traditional neural point process's drawback in capturing the dynamics of the point process governed by multimodal or non-smooth current influence function. Here, we simulate three complicated synthetic datasets using Ogata's thinning algorithm (Chen, 2016). The first is the **Shaky Hawkes process**, characterized by the conditional intensity function

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

where μ, α, and β are scalar parameters that follow the same definition as in an ordinary Hawkes process. Compared to a regular Hawkes process, the Shaky Hawkes process assumes each event's influence function exhibits a cyclic pattern, making it multimodal when no event arrives in a period of time. The second is the **Shift Hawkes process**, characterized by

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

where γ is a scalar threshold value. The shift Hawkes process assumes the influence of each event arrival to be delayed for time γ. The intensity function therefore may have discontinuity between events. The last is the **Delay Peak process**, characterized by

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

Delay Peak process assumes events' influence at arrival is 0. Then the influence gradually emerges like climbing a hill, and finally decreases like going down a hill. The current influence features no change when an event arrives and non-smoothness when it fully disappears. We also include a real-world dataset, earthquake JP, that include the time of all earthquakes in Japan from 1990 to 2020 with magnitude of at least 2.5, gathered by Chen et al. (2020).

4.2 BASELINES

For validating our model's capability of accurately capture functions, in addition to the negative log-likelihood, we include each model's learned intensity's mean absolute percentage error with respect to the ground truth intensity.

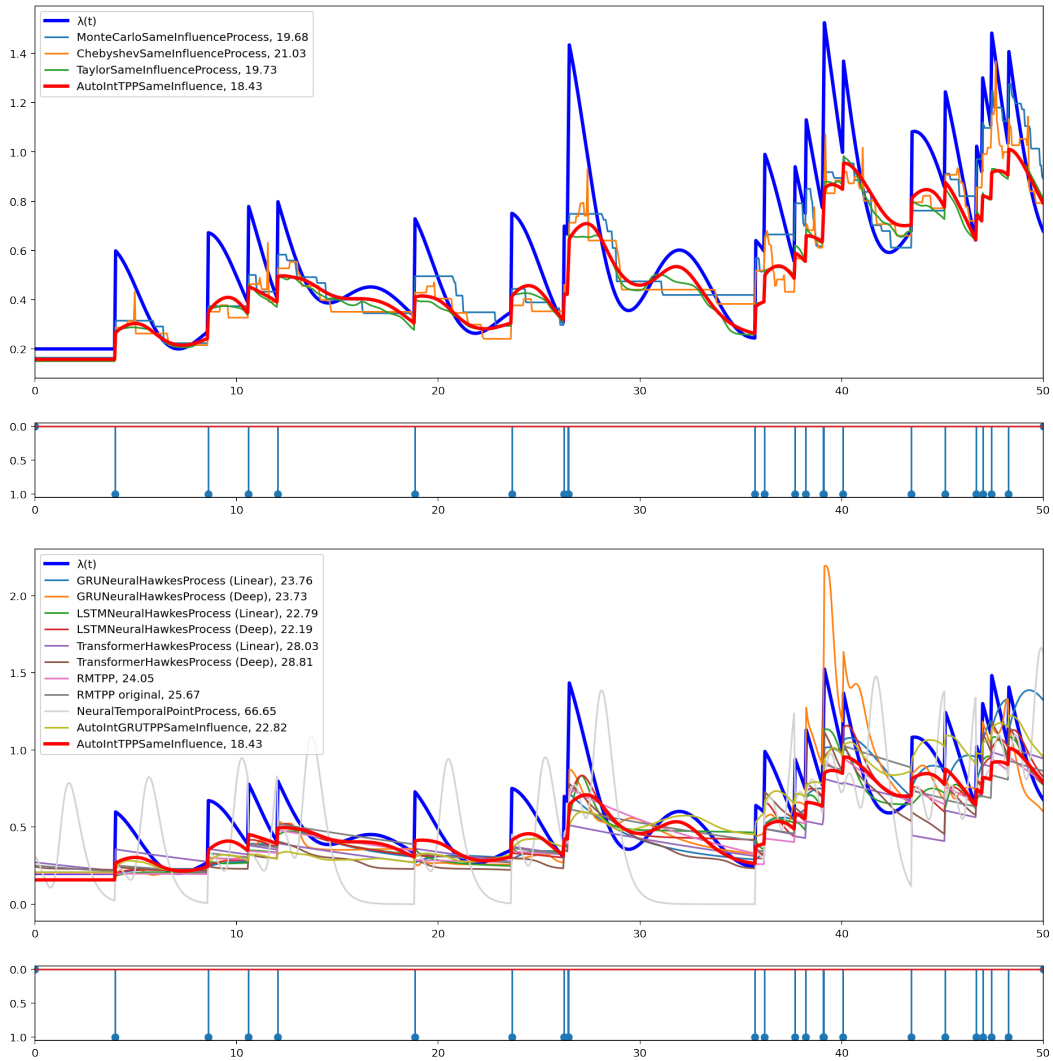
We have two groups of baselines: the first group follows the same model as our setting 1 (that is, to assume all events' influence function is of the same form) but applies different numerical integration techniques. This group of baselines include Liu (2020)'s Taylor approximation method A.1, the Clenshaw–Curtis quadrature method **??** and the Monte Carlo integration method. The first two methods approximate the neural network with orthogonal polynomials (Taylor and Chebyshev

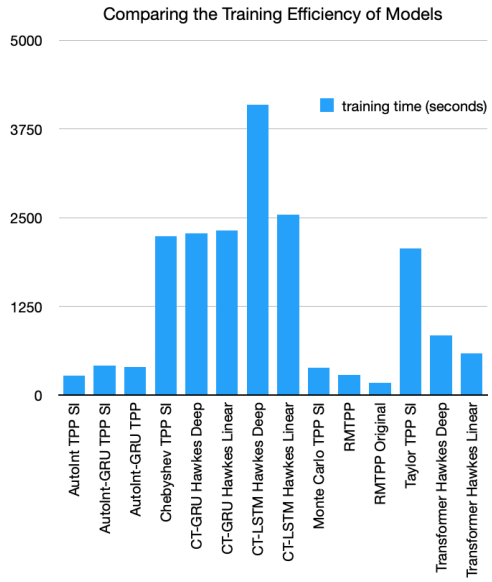
polynomials) and then rely on the easy integrability of polynomials. The third one is the most common method used for modeling the temporal point process.

In the second group, we include state-of-the-art methods with outstanding performance, including RMTPP (Du et al., 2016), Neural-Hawkes (Mei & Eisner, 2016) and Transformer Hawkes (Zuo et al., 2020). Additionally, Mozer et al. (2017) has proposed a continuous-time GRU that interpolates hidden states between events. Its logic is pretty much the same as Neural-Hawkes's continuous-time LSTM. So given that Neural-Hawkes is the only baseline that interpolates hidden states, we additionally include a CT-GRU variant of Neural-Hawkes to increase the diversity of our baselines.

RMTPP is proposed quite early; its current influence function is an exponential decay function and the decay rate is a scalar parameter across all sequences. In order to have a fairer comparison, we added a revised version of RMTPP that has its decay rate being also dependent on the hidden states. We also keep the original RMTPP and call it RMTPP original. Most baseline methods originally use a linear mapping from the hidden state to the intensity function. Given that our inter-event dynamic is complicated, we added "deep mapping" variants of CT-GRU, CT-LSTM Neural-Hawkes, and Transformer Hawkes, which has a three-layer MLP mapping from the hidden states to the intensity function. By adding all these variants, we believe we have a fair coverage of possible TPP baselines.

In the second group, we include state-of-the-art methods with outstanding performance, including RMTPP (Du et al., 2016), neural-Hawkes (Mei & Eisner, 2016) and Transformer Hawkes (Zuo et al., 2020).





Model	shakyHawkes		shiftHawkes		decayPeak		earthquakesJP
	MAPE	LL	MAPE	LL	MAPE	LL	LL
AutoInt TPP SI	**0.1843**	-35.3762	**0.0356**	-39.3599	**0.0373**	-41.9944	8.6187
AutoInt-GRU TPP SI	0.3353	-37.9182	0.4675	-44.0076	0.1107	-42.3124	7.7730
AutoInt-GRU TPP	0.6435	-64.7072	0.4522	-58.4046	0.4100	-65.0632	-2.6385
Chebyshev TPP SI	0.2197	-35.5183	0.0541	-39.4831	17.4032	-451.6057	8.4299
CT-GRU Hawkes Deep	0.2432	-36.3658	0.1235	-39.6273	0.0708	-42.1338	8.6817
CT-GRU Hawkes Linear	0.2243	-35.6063	0.1262	-39.7173	0.1103	-42.1959	5.9148
CT-LSTM Hawkes Deep	0.1938	**-35.2895**	0.1381	-39.8087	0.1623	-42.6656	**9.7533**
CT-LSTM Hawkes Linear	0.2168	-35.4043	0.1473	-40.0411	0.1468	-42.5548	7.0620
Monte Carlo TPP SI	0.1935	-35.6090	0.0462	**-39.3527**	0.0378	**-41.9868**	8.1906
RMTPP	0.2216	-35.4175	0.1515	-39.8077	0.1660	-42.5876	4.8615
RMTPP Original	0.2562	-35.6549	0.2630	-39.7893	0.2183	-42.7965	7.7663
Taylor TPP SI	0.2004	-35.3771	0.0999	-39.7062	0.4674	-45.4099	7.7142
Transformer Hawkes Deep	0.2639	-36.3195	0.2105	-40.2917	0.1864	-42.9732	3.8893
Transformer Hawkes Linear	0.2812	-36.1831	0.2316	-40.6717	0.2342	-43.3308	6.0588

REFERENCES

Ricky TQ Chen, Yulia Rubanova, Jesse Bettencourt, and David Duvenaud. Neural ordinary differential equations. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, pp. 6572–6583, 2018.

Ricky TQ Chen, Brandon Amos, and Maximilian Nickel. Neural spatio-temporal point processes. *arXiv preprint arXiv:2011.04583*, 2020.

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

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

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

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

Nan Du, Hanjun Dai, Rakshit Trivedi, Utkarsh Upadhyay, Manuel Gomez-Rodriguez, and Le Song. Recurrent marked temporal point processes: Embedding event history to vector. In *Proceedings*

of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining, pp. 1555–1564, 2016.

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

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

David B Lindell, Julien NP Martel, and Gordon Wetzstein. 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.

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

Xingchao Liu, Xing Han, Na Zhang, and Qiang Liu. Certified monotonic neural networks. *arXiv preprint arXiv:2011.10219*, 2020.

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

Michael C Mozer, Denis Kazakov, and Robert V Lindsey. Discrete event, continuous time rnns. *arXiv preprint arXiv:1710.04110*, 2017.

Giambattista Parascandolo, Heikki Huttunen, and Tuomas Virtanen. Taming the waves: sine as activation function in deep neural networks. 2016.

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

Oleksandr Shchur, Marin Biloš, and Stephan Günnemann. Intensity-free learning of temporal point processes. *arXiv preprint arXiv:1909.12127*, 2019.

Joseph Sill. Monotonic networks. 1998.

Gregory H Teichert, AR Natarajan, A Van der Ven, and Krishna Garikipati. Machine learning materials physics: Integrable deep neural networks enable scale bridging by learning free energy functions. *Computer Methods in Applied Mechanics and Engineering*, 353:201–216, 2019.

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*, pp. 5998–6008, 2017.

Simiao Zuo, Haoming Jiang, Zichong Li, Tuo Zhao, and Hongyuan Zha. Transformer hawkes process. In *International Conference on Machine Learning*, pp. 11692–11702. PMLR, 2020.

A Appendix

A.1 Taylor Automatic Integration

Let $\beta_1, \ldots, \beta_5$ be real constants with $\beta_1 \neq 0$. 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

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

Then function $\int_a^b f(x)$ can be approximated by $\sum_k \int_a^b y_k(x) A_k$.

A.2 Chebyshev Automatic Integration

We are going to decompose a function $f(x)$ to the sum of the 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, $y = f(x_k)$. 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

$$\int T_n(x)dx = \int T_n(\cos\theta) d\cos\theta \tag{5}$$
$$= -\int \cos(n\theta)\sin\theta d\theta \tag{6}$$
$$= -\frac{1}{2}\int (\sin((n+1)\theta) - \sin((n-1)\theta))d\theta \tag{7}$$
$$= \frac{1}{2}\left(\frac{\cos((n+1)\theta)}{n+1} - \frac{\cos((n-1)\theta)}{n-1}\right) + \text{const.} \tag{8}$$
$$= \frac{1}{2}\left(\frac{T_{n+1}(x)}{n+1} - \frac{T_{n-1}(x)}{n-1}\right) + \text{const} \tag{9}$$

.