Applied Intelligence
https://doi.org/10.1007/s10489-021-02590-1

Learning general temporal point processes based on dynamic weight generation

固定形式的强度函数会限制TPP，现在都用深度TPP

Sishun Liu[1] · **Li Li**[1]

涉及了一个矩阵乘法的模型

Accepted: 2 June 2021

Abstract

poisson process 假设历史事件独立
hawkes process 假设历史事件有激励作用

Most real-world events relate to historical cases scholastically and chronologically. Some increase the probability of the next event in the future, while some do not. Many approaches leverage temporal point processes with explicitly defined probability intensity functions concerning time t and history $\mathcal{H}$, such as Poisson Process and Hawkes Process, to measure the time-varying probability. However, fixed-form intensity functions can limit the performance of temporal point process models, owing to the lack of prior knowledge of required intensity functions and the complexity of the real world. Neural networks' ability to approximate functions makes the neural temporal point process a promising approach compared with the traditional temporal point process. In the paper, we identify several drawbacks the previous works have in meeting the mathematical constraints before designing a matrix-multiplication-based model to tackle these drawbacks. The experimental results reveal that our model is superior to other neural temporal point processes with better mathematical interpretability and extrapolation capability.

Keywords Temporal point process · Weight generation · Irregularly sampled time series

1 Introduction

Events sequences, such as e-commerce and transport transactions, are common in the real world. Most of them are highly correlated and happening at irregular time intervals. It is natural to use the temporal point process to model these sequences. From temporal point processes perspective, the probability density over time is a composition of a conditional intensity function and its integral. If we choose the correct intensity function, temporal point processes can extrapolate along the entire timeline. Classic temporal point processes, like the Poisson process and Hawkes process, assume a fixed-form intensity function to simplify the calculation procedure of integration. After the emergent development of neural networks, it is natural to transfer modern deep learning into temporal point processes [1–11]. The core idea of these models is to replace human-defined knowledge about intensity functions with neural modules. However, this intuitive idea can lead to non-closed form intensity functions and hurt the result precision [9, 10]. Omi et al. [8] require the FullyNN model to learn the intensity functions by fitting the integral of intensity functions. This approach can estimate any randomly given temporal point processes, but it fails to meet several mathematical requisites.

After a detailed investigation of the existing works and mathematical requisites, we present a novel approach satisfying all the constraints without explicit information about the intensity functions. More concretely, by incorporating history and time into the network (adjusting the layer weight), we achieve optimal trade-offs between fully meeting mathematical requisites and having free-formed intensity functions. The proposed model has the following desirable properties: (1) the intensity functions link up to history and time attributions inherently; (2) the integral and conditional intensity functions are continuous and unbounded; (3) zero inputs will get zero outputs regardless of the parameters.

Our contributions are summarized below:

1. Instead of concatenating history and time representations and multiplying them with the fixed weight matrices, we use the embedding of historical events

✉ Li Li
lily@swu.edu.cn

Sishun Liu
liusishun5@gmail.com

[1] School of Computer and Information Science, Southwest University, Chongqing, China

Published online: 09 July 2021

and time intervals to generate the weight. Then apply them with another non-negative time interval embedding. Based on the matrix multiplication principle, we guarantee our model returns zero when the time interval is zero regardless of all model parameters.

2. We avoid the bounded activation functions, such as $tanh$, and suggest a positive trainable activation function $p\alpha ReLU$(positive $\alpha ReLU$) by modifying the definition of $\alpha ReLU$ [12]. $p\alpha ReLU$ can help our approach achieve better extrapolating characteristics by scaling the input distributions, especially when the inputs are limited.

2 Backgrounds

Notation In the paper, we propose an approach to construct distributions over the given event sequences s. Each sequence contains several historical events and their corresponding time t_i. For arbitrary integers i and j that $0 < i < j < length(s)$, we always obtain $t_i < t_j$. Because the time intervals between two adjacent events are common in the temporal point process, we define $\Delta t_i = t_{i+1} - t_i$ and $\Delta t = t - t_l$ in which t_l is the closest time point to t in the corresponding history sequence.

Unless otherwise indicated, we mark all real numbers using lower case Roman letters like x and all vectors using lower case bold Roman letters such as $\mathbf{x}$. All matrices are denoted by capital bold Roman letters like $\mathbf{X}$. An item having an index i in vector $\mathbf{x}$ is denoted as $\mathbf{x}_i$ and $\mathbf{X}_{ij}$ for an item located in i-th row and j-th column in a matrix $\mathbf{X}$. As for functions, we use lowercase Roman letters representing functions returning real numbers. Lowercase bold letters represent functions returning vectors or two-dimensional matrices in which one dimension is 1, and capital-bold letters represent functions returning matrices.

Temporal point process The temporal point process(TPP) is defined over continuous time. Our model can learn it from a strictly increasing time sequence $\mathcal{T} = \{t_1, t_2, \ldots, t_n\}$ - which implies we can acquire the accurate probability value at any time point t in $[t_1, +\infty]$ theoretically. To achieve this goal temporal point process divides its conditional probability distribution into two parts: the probability an event would not happen in the time interval $[t_l, t]$ and the probability an event would in the time interval $[t, t + dt]$. In general, the probability distribution can be characterized by the following equation:

$$P(event\ happens\ in\ [t, t + dt]|\mathcal{H}) = \lambda(t|\mathcal{H})dt \tag{1}$$

In (1), dt is the infinitesimal of t, and $\lambda(t|\mathcal{H})$ is the core of the temporal point process, namely conditional intensity function, or intensity function for short. With this given intensity function, the probability likelihood is defined next as did in [1]

$$p(t|\mathcal{H}) = \lambda(t|\mathcal{H})exp\left(-\int_{t_l}^{t} \lambda(\tau|\mathcal{H})d\tau\right) \tag{2}$$

For simplicity, we use $\lambda(t)$ as shorthand for $\lambda(t|\mathcal{H})$ in the rest of this paper.

Learn intensity functions implicitly Neural networks gain strong function fitness ability from complex hidden representations and various matrix calculations. Du et al. [1] and Zuo et al. [3] introduce RNN and Transformers into temporal time processes by combining history representations and classic intensity functions. It turns that the intensity function helps sequence models to understand time more explicitly.

Although we can directly replace these explicitly defined intensity functions with various neural networks, these integrals in (2) demand expensive Monte Carlo integration. Some researchers seek to obtain more accurate, fast, and interpretable integral by a neural differential equation [13–16], while others ignore intensity functions by modeling conditional probability directly [4, 9]. Omi et al. [8] notice that the probability density function can be regarded as a combination of a function $\Lambda(t) = \int_{t_i}^{t} \lambda(\tau)d\tau$ and its derivative. So instead of modeling intensity function $\lambda(t)$, FullyNN approximates the integral $\Lambda(t)$ using positive neural networks. Thanks to back-propagation, it is sufficient to get the derivative value $\frac{d\Lambda(t)}{dt}$. This algorithm can theoretically learn nearly all intensity functions quite well. Unfortunately, it fails to respect the mathematical constraints. They are: (a) The integral and intensity functions are unbounded. (b) The integration should always return zero when the relative time $t - t_i$ is zero [9].

Explicit encode inputs into weights How to understand and interpret modern Neural Networks (NNs) is a vital concern, and many efforts have been made to explore the interpretation of NNs from various perspectives [17–21]. One of these hypotheses is regarding a NN as an information learner, and information is compressed, stored, and retrievable from the weights of the NN [22–24]. The plainest NN module is the matrix-multiplication-based fully connection layer(FCL), and it can also keep plenty of information, like a 5D neural radiance field [25]. What's more, matrix multiplication can comfortably meet both constraints (a) and (b). Unfortunately, we cannot directly utilize FCL because the historical information and the temporal information can not be dealt with simultaneously and harmoniously in the traditional FCL.

In recent years, Transformer-based models [26–28] show that it is better to generate linear transformation matrices by input and model parameters instead of input-irrelevant ones. Inspired by this idea, we change the definition of normal

FCL by injecting input into its weight matrix. Detailed discussions about why vanilla FCL fails and how to modify it are present in Section 3.1.

3 Method

From what elaborated on above, the requisites our model should satisfy are as follows:

1. The function should be monotonically increasing because the intensity function is related to conditional probability, which is positive.
2. The function and its derivative are continuous and not bounded when $t \geqslant 0$.
3. The function should always return zero when the relative time gap $t - t_l$ is zero.

After careful investigations, we found the third one is the most tricky. Most neural network constraints are implemented by somehow constrained parameters [8, 29, 30], activation functions [31], loss functions [32, 33], etc. However, none of them works if one suppose to force DNN models to return a given constant number at a given point t regardless of the model parameters. One way we found to handle this is to calculate the approximations at t_l, and offset the original estimator outputs with the following expression [10]:

$$\Lambda(t) = ReLU(\Lambda^*(t) - \Lambda^*(t_l)) + \epsilon \tag{3}$$

where ϵ is a small positive value, $\Lambda^*(\cdot)$ refers to the learned function. In the "comparison with FullyNN" subsection, we will use the above compromising method to implement a zero-integral FullyNN model.

3.1 Dynamic weight layer

Our approach is inspired by a common multiplication principle.

$$x * 0 = 0 \tag{4}$$

where x refers to an arbitrary real number. If we replace x with a vector $\mathbf{x}$ or matrix $\mathbf{X}$ and 0 into a corresponding zero vector or a zero matrix, the result is $\mathbf{0}$. Considering it under the given integral circumstance, we rewrite (4) into the following equation.

$$\mathbf{W} * \mathbf{p}(t_i - t_i) = \mathbf{0} \tag{5}$$

where $\mathbf{W}$ refers to model parameters, $\mathbf{p}(\cdot)$ is a monotonic increasing projection function to project a given number into a vector. $\mathbf{p}(\cdot)$ should consist of matrix multiplications.

Equation 5 illuminates a possible path to satisfy the third requisite. But soon, we discover it is impossible to inject history information into time embedding by projection function $\mathbf{p}(\cdot)$ directly because neither the history embedding nor the weight values relevant to history can be $\mathbf{0}$. Otherwise, our approach loses all historical information. What's more, these non-zero history embeddings violate (5) then cause models to fail on the third requisite.

The above analysis suggests it is better to move historical information into $\mathbf{W}$ instead of t. However, another issue hides in the derivative of the commonly used $softplus$ activation function when generating history-related-only weight matrices. Although the $softplus$ activation itself doesn't have upper bounds, the derivative of $softplus$ has a supremum $y = 1$. This bounded derivative will limit the intensity function values no greater than the product of weight matrices $\mathbf{W}$ in full-connection layers. As a result, the models are willing to approximate constant intensity values when given great t. Further experiment results proving this elaboration are present in Section 4.

According to the above analysis, we change the definition of $\mathbf{W}$ from a fixed weight matrix into a dynamic generated weight matrix $\mathbf{W}(\mathcal{H}, t, t_l)$ to ensure:

1. The derivative of integral $\Lambda(t)$ directly contains $\mathbf{W}(\mathcal{H}, t, t_l)$. It indicates the intensity function $\lambda(t)$ preserves history $\mathcal{H}$ and time t.
2. The form of matrix multiplication is maintained while (5) still holds.
3. The weight $\mathbf{W}$ can reach infinity by accumulating monotonic time embedding to avoid finite derivatives introduced by the $softplus$ activation function.

Finally, we obtain the formulation of dynamic weight layer:

$$L(\mathcal{H}, t, t_l) = \mathbf{W}(\mathcal{H}, t, t_l) * \mathbf{p}(t - t_l) \tag{6}$$

3.1.1 Layer implementation

$$\mathbf{H_w} = LSTM(\mathcal{H}) \tag{7}$$

$$\mathbf{T_w} = \mathbf{p_1}(TA(p\alpha ReLU(t - t_l; a))) \tag{8}$$

$$\mathbf{W} = softplus\left((\mathbf{T_w} + \mathbf{H_w})\mathbf{W_g}\right) \tag{9}$$

$$\mathbf{H} = \mathbf{W}^T * \mathbf{p_2}(t - t_l) \tag{10}$$

$$p\alpha ReLU(x; a) = softplus(a) * ReLU(x) \tag{11}$$

The actual approach of generating $\mathbf{W}$ in the paper is listed in (7), (8), and (9). Firstly, we map the history and time interval into $\mathbf{H_w} \in \mathbb{R}^{H\times 1}$ and $\mathbf{T_w} \in \mathbb{R}^{H\times 1}$ using an LSTM and a linear module, respectively. In the time interval mapping layer, we apply $p\alpha ReLU$ to the time interval firstly. After blending these two representations, we map history and time representation $\mathbf{T_w} + \mathbf{H_w}$ to the needed dynamic weight matrix $\mathbf{W} \in \mathbb{R}^{H\times I}$ by another matrix multiplication with $\mathbf{W_g} \in \mathbb{R}^{1\times I}$. Positive activation $softplus$ is applied to preserve this layer yielding a positive

gradient. It is concerned that it may lead to a zero $\mathbf{W}$ matrix if all original values in $(\mathbf{T_w} + \mathbf{H_w})\mathbf{W_g}$ are negative infinity when we feed in infinite t. We find this condition is rare. During the entire experiment, we don't observe a single case of such a circumstance.

The $p\alpha ReLU$ applied here is reasonable. The self-exciting temporal point process's intensity value decreases and finally very close to base intensity a_i, which means greater time intervals have nearly identical hidden representations. On the other hand, the self-correcting process has a drastic increasing intensity function, indicating these big-time intervals mapping to bigger embedding. Facing such different requirements, time activation $TA(\cdot)$ and the scaling factor in $p\alpha ReLU$ help models handle the input distributions to better approximating various intensity functions. In the experiment section, we will evaluate the effect of different $TA(\cdot)$ and $p\alpha ReLU$.

After the weight generation part, the remains of this layer is a simple matrix multiplication between $\mathbf{W} \in \mathbb{R}^{H\times I}$ and time gap representation vector $\mathbf{T} = \mathbf{p_2}(t - t_l) \in \mathbb{R}^{H\times 1}$.

3.2 The model

$$\mathbf{H}_{out} = p\alpha ReLU(\mathbf{W}_p * \mathbf{H}_{in}; b)(\mathbf{W}_p \geqslant \mathbf{0}) \tag{12}$$

$$o = FA\left(\mathbf{W}_H^T * \mathbf{H}_{out}\right) \tag{13}$$

The remaining structure of the proposed model is similar to FullyNN [8]. We use $p\alpha ReLU$ equipped with another trainable parameter, b. Among all non-negative layers, $p\alpha ReLU$s share parameter b. After several non-negative dense layers ((12) refers to a single layer), $\mathbf{H}_{out} \in \mathbb{R}^{I\times 1}$ multiplies $\mathbf{W}_H^T \in \mathbb{R}^{1\times I}$ to get integral outputs. Figure 1 shows the model architecture conceptually.

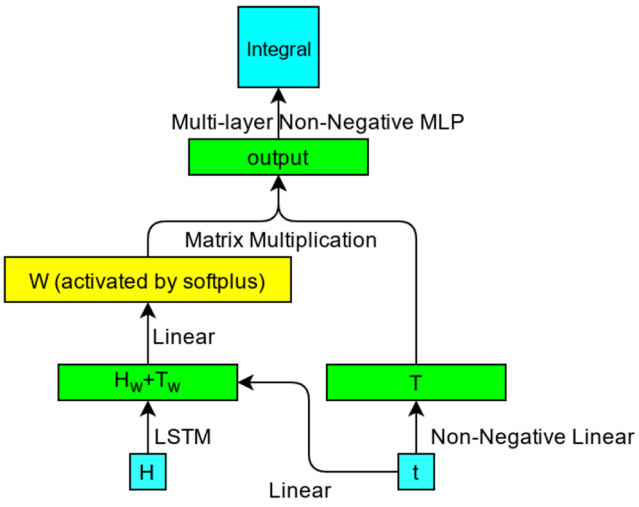


Fig. 1 The model architecture

Loss function We follow the MLE method to train proposed models by using mean negative log-likelihood as the loss function.

$$l(t) = -\log(\lambda(t)) + \int_{t_l}^{t} \lambda(\tau)d\tau \tag{14}$$

Following the given definition, we obtain that a lower loss may not refer to a better model. For example, in the static Poisson process, the intensity function $\lambda(t)$ is 1, and integral $\Lambda(t)$ is t. We can obtain the ideal loss is $t - t_l$, not 0. Based on our observation, we provide two results in all MNLL loss charts: absolute MNLL loss l and relative MNLL loss $l_r = l - l_{ideal}$. We denote them in form l/l_r, and better models should have lower relative MNLL loss values. Real-world datasets don't have ground truth intensity functions. Thus, we use MAE(Mean Absolute Error) between ground truth and time predictions to judge the model performance.

We also notice our models continuously get lower MNLL loss with epoch numbers increasing, making MNLL losses unconvincing when MNLL losses(both absolute and relative MNLL loss) among models are close to each other. In the paper, we choose to highlight lower relative loss values while emphasizing that it works well only with a notable loss gap.

Baseline settings Unless otherwise indicated, we use the following settings for all training processes.

- Batch size, epoch, and optimizer: Among all models, the batch size is 512 by default. It is an empirical choice. Smaller or bigger batch sizes may influence the performance, but we think it is minor. We use AdamW [34] as our default optimizer. Unless specified, we train all models with 100 epochs, and the default learning rate is 0.1.
- Dimensions: We have two dimension setups: 32 for both history embedding and intensity embedding, or 32 for history embedding, 64 for intensity embedding.
- RNN and MLP layers: We have four different combinations of layer counts: (a) 1 LSTM and 2 MLPs. (b) 1 LSTM and 3 MLPs. (c) 1 LSTM and 4 MLPs (d) 2 LSTMs and 3 MLPs. No normalization is applied. Additionally, we insert one dynamic weight layer between LSTM and MLPs and don't count it as an MLP layer. To maintain the same model configurations between our model and FullyNN, we compensate an MLP layer to all FullyNN models.
- Initialization: We apply the same random seed in all training procedures. Unless specified, initial trainable parameters in all $p\alpha ReLU$ are $softplus(0) = log(2) \approx 0.693$.

3.3 The relation between proposed method and spiking neural network

Spiking neural network(SNN) is a biologically plausible neural network, and it is used for robot control [35–37], computer vision [38, 39], sequence learning [40–42], etc. The salient feature of SNN is that its neurons only excite and send a spiking event when the inner membrane potential exceeds a spiking threshold. From the mechanism perspective, the proposed method is similar to spiking neural networks. However, the differences between them are still noticeable.

1. A spiking neuron always has a threshold, and it sends a spike as soon as its membrane potential reaches the threshold [43–45]. The temporal point process doesn't have an upper bound because the integral turns to positive infinity if we want $P(x|H) = 1$. Our approach ensures maximizing the integration as much as possible when an event happens, and we argue similar threshold design from SNN can not be introduced into our approach.
2. A spiking neuron resets itself after sending a spiking event, and then a refractory period applies [44, 45]. As for the temporal point process, there is no refractory period after an event happens. That means we can get great event density.
3. Spiking neural networks are inspired by the human brain. The training methods of spiking neural networks are different from that of artificial neural networks(ANNs). For SNN, Hebbian-based Spike-Timing-Dependent Plasticity(STDP) mechanism and gradient descent are widely-used training techniques [46, 47]. Our proposed method doesn't fit the STDP mechanism because our approach gets derivative values of $\Lambda(t)$ via backpropagation. It indicates that our model can only be trained by gradient descent.
4. Spiking neural networks always hold a discontinuous function [44]. Meanwhile, our approach can not handle these instant shocks spontaneously because our model is continuous.

The general temporal point process and spiking neural networks are different from each other. Because both definitions contain differential equations [1, 44], we are aware of the similarity between our approach and spiking neural networks.

Spiking neural networks can produce more biological dynamics. If one combines spiking neurons to mimic neuro-biological architectures, such as the brain, these architectures can be more convincing and powerful than classic VLSI-based ones. This model design concept, called digital neuromorphic computing, has directed various existing works towards robots, cognition tasks, and real-time control [47–49]. Due to the similarity between the proposed model and spiking neural network on the definition and differential equation utility, we hope this work can also contribute to spiking neuron network in general, implicit evolution function learning. Thus, it may improve the digital neuromorphic computing capability.

4 Experiment

We explore our model in several different ways on synthetic data and real-world data. Since the main target of the model is to solve the known issues in FullyNN, we not only compare the performance between our model and FullyNN, but we also show how FullyNN suffers from these issues and how well our model can tackle them.

4.1 Synthetic data

In the paper, we generate five different synthetic datasets, as described in Omi et al. [8], to evaluate our model and other temporal point process models.

Homogeneous poisson process Homogeneous Poisson process a simple temporal point process. Its intensity function is a constant, namely $\lambda(t) = c$. In the paper, we set $c = 1$.

Hawkes process Hawkes process, one of the self-exciting processes, has a time-related intensity function: $\lambda(t) = c_0 + \sum_{t_i<t} \sum_{j=1}^{n} a_j b_j exp(-b_j(t - t_i))$. Here we generate two different synthetic datasets: Hawkes_1 with $n = 1, c_0 = 0.2, a_1 = 0.8, b_1 = 1.0$ and Hawkes_2 with $n = 2, c_0 = 0.2, a_1 = 0.4, b_1 = 1.0, a_2 = 0.4, b_2 = 20.0$, respectively.

Self-correcting process Contrary to the self-exciting process, the self-correcting process has a monotonic increasing intensity function. The intensity function is $\lambda(t) = exp(t - \sum_{t_i<t} 1)$ in the experiments.

Stationary-renewal process In this process, the intervals Δt are randomly distributed from a given probability distribution. In other words, there might have either many events or no event that a single one ensues. In the paper, we use a log-normal distribution to generate time intervals.

4.2 Real world data

We use the following four real-world datasets to validate our model.

Subreddit[1] The original dataset contains 14 million records of comments posting in 34,967 subreddits. In the paper, we choose comments posted in two subreddit, namely *nfl* and *funny*. There are 173,883 notes in *nfl* and 152,921 in *funny*, respectively. Because of the rapid growth of these communities, we notice that the average and variance of time intervals change dramatically along time, reflecting a probability distribution variation. To mitigate this issue, we choose a part of each given dataset and apply time scaling(*nfl*:last 121,000 logs with 0.015 scalings; *funny*: last 102,000 logs with 0.005 scalings).

Earthquake[2] The dataset contains 24,007 earthquake information in Turkey from 1910 to 2017. The probability of an earthquake is influenced by geographical location. Therefore we don't choose global earthquake information, and we believe a global one is more feasible for the marked temporal point process(MTPP). The time scaling factor is 0.01. We expect a greater MAE loss since the time interval average is much bigger than all others(23 v.s. 1).

LastFM [1] This dataset contains 929 user event sequences from LastFM. The average length of these sequences is 1,365. After preprocessing the raw dataset into history-target pairs, we choose the first 153,600 records as the training set and the other 25,600 as the test set. The average time interval is 0.47, so we set the time scaling factor as 5. To stabilize the training procedure, we place a lower bound to weight matrices in $\mathbf{p_1}$ and $\mathbf{W_g}$. For this particular dataset, the lower bound is -0.15.

Retweet [50] This dataset contains sequences of retweets from October 7 to November 7, 2011. For the sake of collecting more historical information, only tweets having at least 50 retweets are recorded. The original dataset has 166,076 tweets and over 34 million records. We select 800 tweets(233,005 records) for preprocessing. The time scaling factor is 0.005. Finally, the initial value of time scaling factor a is $softplus(10)$.

Detailed hyperparameter settings about training our models are described in Appendix Section 1.1. The reason and influence of increasing the initial value of a and applying weights lower bounds are reported in Appendix Sections 1.2 and 1.3, respectively.

[1] https://www.kaggle.com/colemaclean/subreddit-interactions

[2] https://www.kaggle.com/caganseval/earthquake

5 Result and discussion

5.1 Ablation experiment

In this section, we investigate our model under multiple different settings to find out how each part influences the model performance. Experiment metrics are mean negative log-likelihood(MNLL) and mean relative negative log-likelihood(relative MNLL). The definition of relative log-likelihood is the subtraction between MNLL and MNLL truth. The revealed results are the mean of results among different hyperparameters, and lower relative MNLL is better. For prediction, the predicted period for intensity and integral values have the same history within the time interval [0, 10]. All used models for plotting share the same hyperparameters(100 epoch, 1 LSTM layer, 3 MLP layers, 32-d history embedding, 64-d intensity embedding, 512 batch size).

5.1.1 Weight generation

The original idea suggests we should generate weight $\mathbf{W}$ with both history $\mathcal{H}$ and input time t. Otherwise, the model may suffer the limited intensity issue. Table 1 shows the comparison on "with and without time embedding" Fig. 2 shows the predicted integral and intensity functions.

Interestingly, we observe the output activation function $FA(\cdot)$ in (13) restricts the result of time-irrelevant models. The core inducement is gradients can only receive time information from $FA(\cdot)$, whose derivative is fixed. It forces the time-irrelevant models to fit various intensity functions in a fixed form, and that's why distributions time-irrelevant models predict are akin. Baselines' weight matrix $\mathbf{W}$ is time-relevant. Therefore, it can fit different intensity functions better than time-irrelevant ones.

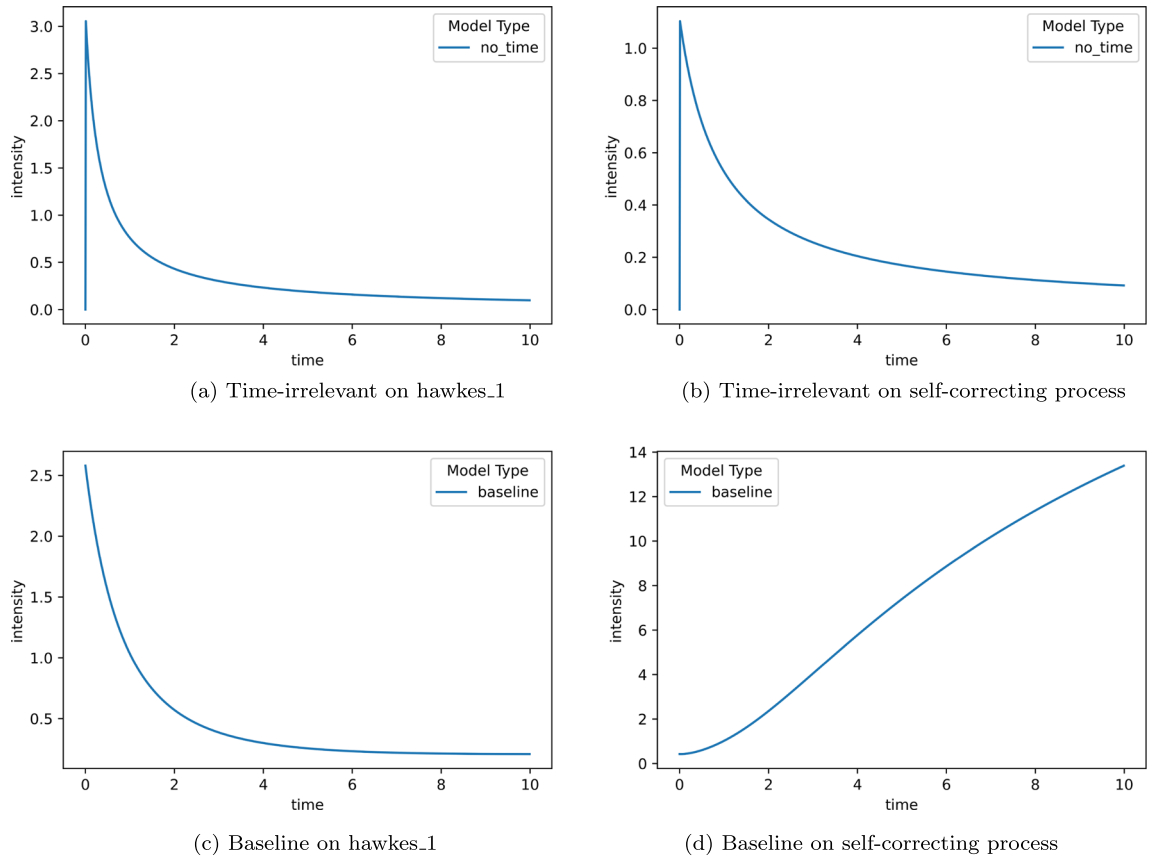
5.1.2 Activation function

We set $p\alpha ReLU$ in our model and compare it with standard $ReLU$ activation. Results are listed in Table 2.

The reason for using trainable activation functions is to allow models to find appropriate input intervals to match other fixed activation functions. It is natural to hypothesize the method works reasonably well if the inputs are limited by the activation functions and required intensity functions fluctuate drastically over time. In the current subsection, we revisit this hypothesis by limiting time interval Δt with $tanh$ in the weight generation process. From the results listed in Table 2, we obtain this method works better than normal $ReLU$ in all datasets except Poisson since the

Table 1 Performance comparisons between time-relevant weight generations

	Hawkes_1	Hawkes_2	Poisson	Self-correcting	Stationary renewal
Time-irrelevant	0.562/0.112	0.159/0.082	1.116/0.137	1.179/0.434	0.241/0.032
Baseline	0.521/0.072	0.060/-0.010	0.978/-0.005	0.771/0.026	0.228/0.021



(a) Time-irrelevant on hawkes_1

(b) Time-irrelevant on self-correcting process

(c) Baseline on hawkes_1

(d) Baseline on self-correcting process

Fig. 2 Time-irrelevant model (a, b) and baseline model (c, d) intensity prediction on Hawkes_1 and Poisson dataset. It shows that time-irrelevant models are significantly affected by the final activation function, namely $Log(x) = log(1 + x)$, while baseline models are not

Table 2 Performance comparisons between trainable positive activation functions

	Hawkes_1	Hawkes_2	Poisson	Self-correcting	Stationary renewal
Tanh with ReLU	0.587/0.156	0.173/0.100	0.978/-0.005	0.808/0.063	0.315/0.105
Tanh baseline[a]	0.524/0.074	0.159/0.086	0.978/-0.005	0.770/0.026	0.290/0.080
Baseline with ReLU	0.523/0.074	0.062/-0.006	0.978/-0.005	0.772/0.027	0.230/0.023
Baseline	0.521/0.072	0.060/-0.010	0.978/-0.005	0.771/0.026	0.228/0.021

[a]"Tanh baseline" refers to models displacing time activation function from $Log(x) = log(1 + x)$ to $tanh(x)$ and leaving anything else unchanged

Table 3 Activation parameters in Tanh baseline (The selected model has one LSTM layer, three MLP layers, 32-d history embedding and 64-d intensity embedding)

	Hawkes_1	Hawkes_2	Poisson	Self-correcting	Stationary renewal
Time activation	0.1962	0.7594	0.6915	0.2533	0.5145
Layer activation	0.6676	0.6803	0.6602	0.6711	0.6731

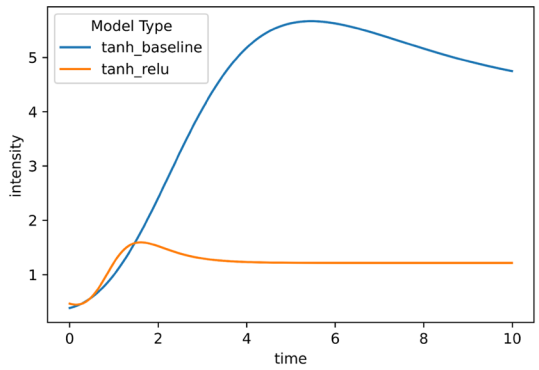
Poisson process is too simple for both models. Table 3, Fig. 3(a) and (b) show the proposed $p\alpha ReLU$ does help the models with limited inputs better fit the distributions by significantly delaying the intensity decay.

As for baseline models, Table 4 shows the values of a and b in one of the trained baseline models, and corresponding prediction curves are shown in Fig. 3(c) and (d). We observe that $p\alpha ReLU$ still plays a role in both the Hawkes_2 process and stationary renewal process. However, as $Log(x) = log(1+x)$ is already unbounded, these highly shifted parameters make fewer influences on final MNLL loss than the Tanh baseline.

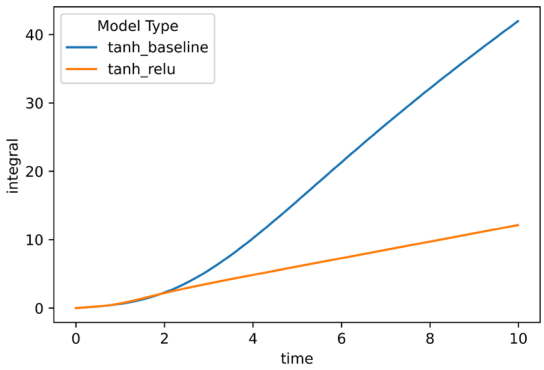
5.1.3 Time activation function

In Table 5, we evaluate two different time activation functions $TA(\cdot)$, namely $Log(x) = log(1+x)$and $tanh(x)$. We check their extrapolation performance by predicting the intensity values on a great time scale. Predicted intensity curves on self-correcting and hawkes_2 are shown in Fig. 4.

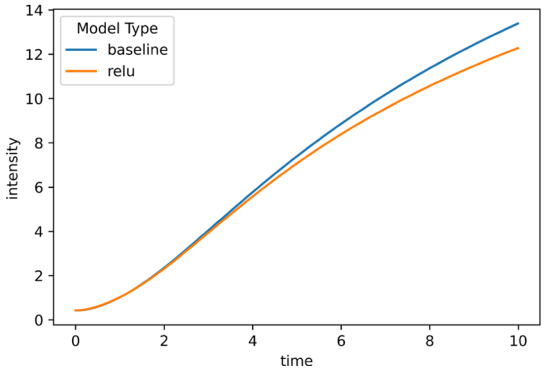
The results indicate that baseline models outperform the Tanh baseline on two synthetic datasets. Tanh baseline models have less extrapolating ability because of the bounded time activation functions. In Fig. 4(a), two curves coincide with each other when t is small. But apparently, the



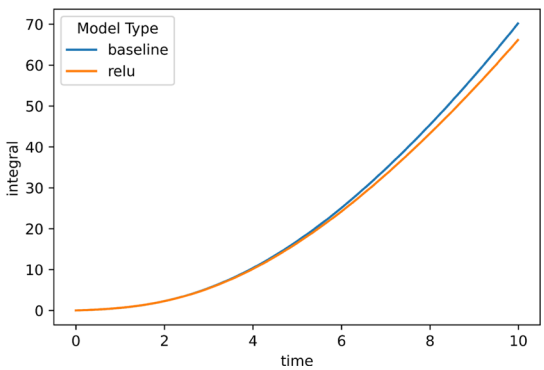
(a) Intensity predictions on self-correcting(Tanh baseline and Tanh baseline with ReLU)



(b) Integral predictions on self-correcting(Tanh baseline and Tanh baseline with ReLU)



(c) Intensity predictions on self-correcting(baseline and baseline with ReLU)



(d) Integral predictions on self-correcting(baseline and baseline with ReLU)

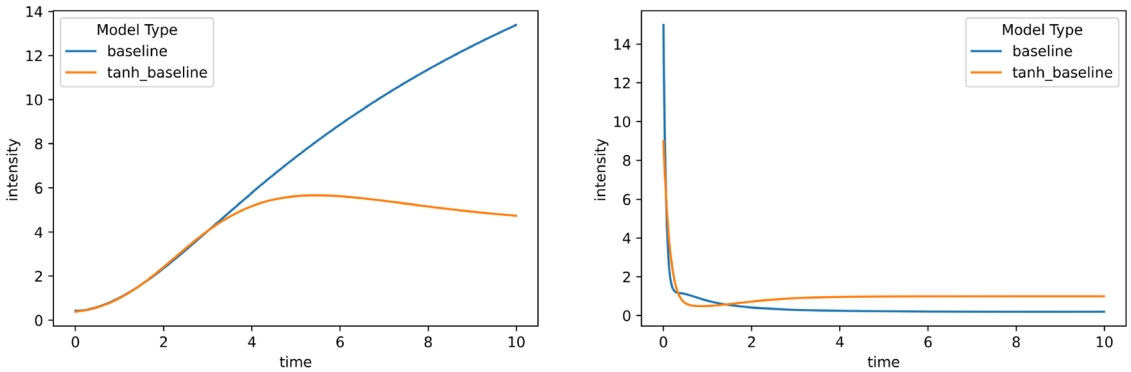
Fig. 3 Performance of activation functions for various models. *tanh* limits the gradient, causing intensity decay. Proposed $p\alpha ReLU$ relieves this tendency to some extent

Table 4 Activation parameters in baseline (The selected model has one LSTM layer, three MLP layers, 32-d history embedding and 64-d intensity embedding)

	Hawkes_1	Hawkes_2	Poisson	Self-correcting	Stationary renewal
Time activation	0.6516	1.9024	0.6777	0.6725	1.5405
Layer activation	0.6667	0.6814	0.6602	0.6707	0.6732

Table 5 Performance comparisons between time activation functions

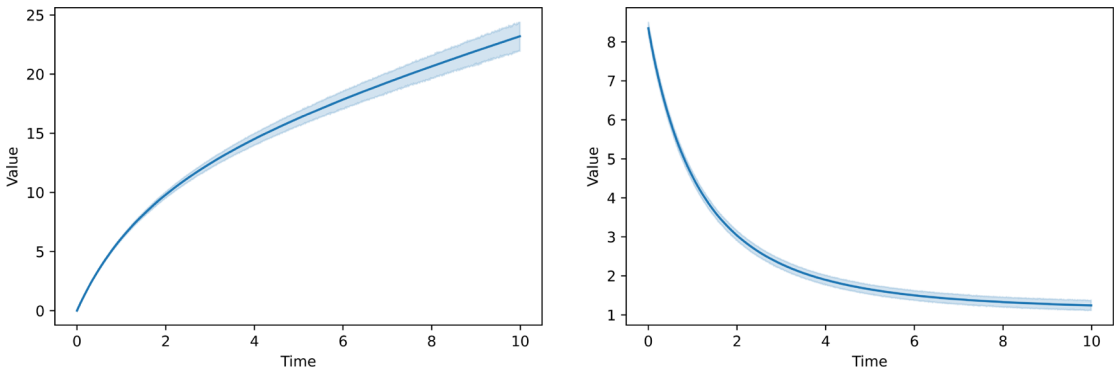
	Hawkes_1	Hawkes_2	Poisson	Self-correcting	Stationary renewal
Tanh baseline	0.524/0.074	0.159/0.086	0.978/-0.005	0.770/0.026	0.290/0.080
baseline	0.521/0.072	0.060/-0.010	0.978/-0.005	0.771/0.026	0.228/0.021



(a) Intensity predictions on self-correcting(baseline and Tanh baseline)

(b) Intensity predictions on hawkes_2(baseline and Tanh baseline)

Fig. 4 Effects of time activation functions for various models. In subplot (a), the Tanh baseline fails to preserve correct intensity tendency as time increasing. In subplot (b), the baseline successfully converges to base intensity $c_0 = 0.2$, while the Tanh baseline does not



(a) Integral predictions on hawkes_1 with different seeds

(b) Intensity predictions on hawkes_1 with different seeds

Fig. 5 The intensity and integral prediction comparison between approaches with different seeds. One can observe that different seeds can only slightly affect the prediction curves(the mean absolute loss of all ten models is 0.521, and the variance is 4.74×10^{-4}). Other synthetic datasets also report the same results, indicating that our approach is robust against random seeds

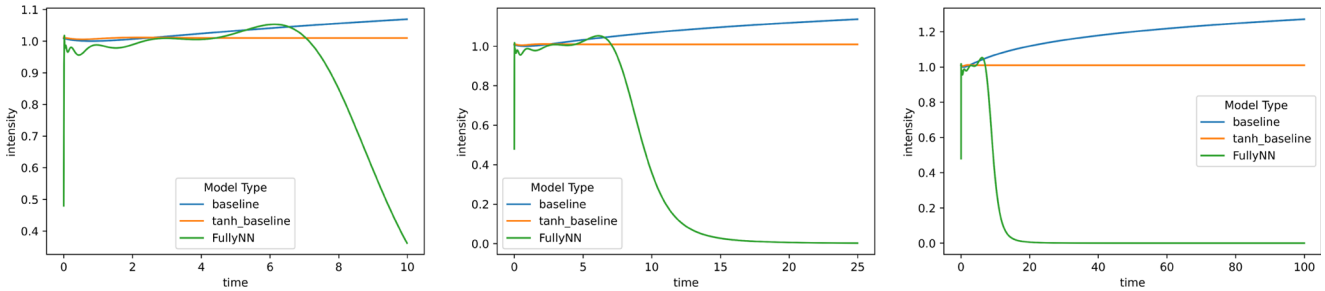


Fig. 6 Extrapolation comparison between our models and FullyNN on Poisson process. The true intensity is $\lambda(t) = 1$. In subplot (a), we observe FullyNN hits the bound of *tanh* and starts decreasing when $t = 6$, while all our models are robust. As the input time becomes greater, FullyNN starts outputting low values since *tanh* limits the integral output, while baseline and Tanh baseline holds the correct intensity function better

Table 6 Performance comparisons with FullyNN on synthetic datasets

	Hawkes_1	Hawkes_2	Poisson	Self-correcting	Stationary Renewal
FullyNN	0.523/0.074	0.067/-0.003	0.979/-0.003	0.771/0.027	0.224/0.020
FullyNN(offset)	0.519/0.071	0.056/-0.012	0.978/-0.005	0.769/0.026	0.226/0.019
FullyNN(200 Epoch)	0.521/0.072	0.062/-0.009	0.979/-0.004	0.770/0.026	0.223/0.019
Tanh baseline	0.524/0.074	0.159/0.086	0.978/-0.005	0.770/0.026	0.290/0.080
Baseline	0.521/0.072	0.060/-0.010	0.978/-0.005	0.771/0.026	0.228/0.021

Table 7 The integral value when $t = 0$

	Hawkes_1	Hawkes_2	Poisson	Self-correcting	Stationary renewal
Our models	0	0	0	0	0
FullyNN	0.0003	0.0013	0.0008	0.0005	0.0005

Table 8 Performance comparisons with FullyNN on real-world datasets (MAE)

	Reddit(nfl)	Reddit(funny)	Earthquake	Retweet	LastFM
FullyNN	0.712	1.080	6.360	7.622	2.084
FullyNN(offset)	0.714	1.080	6.360	7.611	2.087
FullyNN(200 Epoch)	0.714	1.080	6.360	7.614	2.083
Tanh baseline(ours)	0.713	1.080	6.359	–	2.148
Baseline(ours)	0.711	1.080	6.359	7.629	2.080

difference gets significant with the time interval increasing. As for the Hawkes_2 process, Tanh baseline outputs 1 when given a big timestamp while the baseline model is around 0.2. On the other three synthetic datasets, we also observe similar results. It proves that the tanh baseline models have few advantages with great time intervals.

5.1.4 Model robustness

In this section, we explore the training robustness of the proposed approach. Because we have already trained models using different hyperparameters and the loss variance is tiny(smaller than 10^{-3}), the model is robust with various MLP layer numbers, epoch numbers, hidden state dimensions, and LSTM layer numbers. We contemplate if different random seeds make any perturbations to the model. Ten random seeds are generated by a random number generator, and other hyperparameters stay unchanged in the experiments. Figure 5 presents the prediction curves.

From these figures, it is concluded that random seeds have little impact on the results. The reason behind it is that the proposed approach always approximates the correct integral and corresponding intensity function. Random seed only affects how fast the training procedure converges. Thus it proves that our approach is robust.

5.2 Comparison with FullyNN

5.2.1 Comparison on synthetic data

In this section, we evaluate the performance between our model and FullyNN on the given synthetic datasets. As demonstrated above, FullyNN's layer activation function, namely $tanh$, is quite feasible to help model fit intensity functions with long tails like Hawkes intensity function, but the upper bound and lower bound could drastically affect the extrapolating performance of models. In the following experiment, we expand the range of time t from [0, 10] to [0, 25] and [0, 100] to evaluate the extrapolating ability of our models and FullyNN. Since the average time in all training datasets except Earthquake is around 1, requiring reasonable intensity values in [0, 100] needs robust extrapolation characteristic. Figure 6 shows how models act on the Poisson process. MNLL loss comparisons are listed in Table 6. Model values at 0 are reported in Table 7 separately.

From Table 6, we conclude that our model behaves as what FullyNN does in all synthetic datasets. We also observe the impact of $tanh$ activation function on these original FullyNN models in Fig. 6. When the input is reasonable compared with that in the training dataset, both FullyNN and our models can approximate the intensity function nicely. As time t increases, the $tanh$ activation functions between full-connection layers in FullyNN hits the upper bound, causing a rapid gradient drop between layers and, finally, a drastic reduction in intensity prediction values. Meanwhile, Our models are robust thanks to the unbounded weight matrix $\mathbf{W}$ and the direct time scaling technique provided by $p\alpha ReLU$.

Our approach succeeds strictly in respecting $f(0) = 0$, as shown in Table 7. Although FullyNN has small outputs, it is still too hard for FullyNN to firmly meet the required constraint without any explicit hints.

5.2.2 Comparison on real-world data

The comparisons between our models and FullyNN on the real-world datasets are addressed in this section. The MAE results are shown in Table 8. In the paper, the predicted time t_p should meet requirement $P(event\ happens\ in\ [t_p, t_l + t_p]) \geqslant 0.5$. From (2), we obtain it is equivalent to $\int_{t_l}^{t_p} \lambda(\tau)d\tau \geqslant log(2)$. After ascertaining that all trained models are monotonic increasing, we use binary search to find appropriate t_p.

The baseline model fails to converge on several real-world datasets with too many non-negative MLP layers(4 layers) or LSTM layers(2 layers). We find all these models have learned an abnormal value in layer activation functions (over 2). It leads to an unexpected exponential explosion in the results. After removing divergent models, we find our models match FullyNN on all five real-world datasets without bounded predictions and $\int_{t_l}^{t_l} \lambda(\tau)d\tau \neq 0$ issues.

6 Conclusion

In the paper, we design a matrix-multiplication-based novel approach to meet the strict value requisite $f(0) = 0$ and replace $p\alpha ReLU$ with $ReLU$ activation to shift the input time distribution adaptively. Our experiment results show that our method outperforms existing general temporal point process learning methods, such as FullyNN, on extrapolation characteristics and strict constraint obedience. As both properties result in more robust long-time predictions and higher mathematical interpretability, our approach is more competent to solve TPP prediction tasks, especially the strong extrapolation-required ones.

Supplementary Information The online version contains supplementary material available at https://doi.org/10.1007/s10489-021-02590-1.

Acknowledgements This work is funded by the National Natural Science Foundation of China (ID:61877051) and the National Science Foundation Project of CQ (ID: cstc2018jscx-msyb1042 and cstc2018jscx-msybX0273). We thank Senlin Shu and Xiaofei Xu for all the inspiring discussions about the design of the dynamic weight layer. Additionally, we acknowledge all the developers and

researchers for developing useful machine learning tools that enable our experiments.

References

1. Du N, Dai H, Trivedi R, Upadhyay U, Gomez-Rodriguez M, Song L (2016) Recurrent marked temporal point processes. In: Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining. ACM, New York, USA, pp 1555–1564
2. Mei H, Eisner J (2017) The neural hawkes process: A neurally self-modulating multivariate point process. In: Advances in neural information processing systems 30: Annual conference on neural information processing systems 2017, December 4-9, 2017, Long Beach, CA, USA, pp 6754–6764
3. Zuo S, Jiang H, Li Z, Zhao T, Zha H (2020) Transformer Hawkes process. In: Proceedings of the 37th international conference on machine learning, Proceedings of machine learning research, vol 119, pp 11692–11702. PMLR
4. Li S, Xiao S, Zhu S, Du N, Xie Y, Song L (2018) Learning temporal point processes via reinforcement learning. In: Advances in neural information processing systems 31: Annual conference on neural information processing systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal Canada, pp 10804–10814
5. Guo R, Li J, Liu H (2018) INITIATOR: noise-contrastive estimation for marked temporal point process. In: Proceedings of the twenty-seventh international joint conference on artificial intelligence, IJCAI 2018, July 13-19, 2018, Stockholm, Sweden, pp 2191–2197. ijcai.org
6. Okawa M, Iwata T, Kurashima T, Tanaka Y, Toda H, Ueda N (2019) Deep mixture point processes: Spatio-temporal event prediction with rich contextual information. In: Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining, KDD 2019, August 4-8, 2019. ACM, Anchorage, AK, USA, pp 373–383
7. Xiao S, Yan J, Farajtabar M, Song L, Yang X, Zha H (2019) Learning time series associated event sequences with recurrent point process networks. IEEE Trans Neural Netw Learn Syst 30(10):3124–3136
8. Omi T, Ueda N, Aihara K (2019) Fully neural network based model for general temporal point processes. In: Advances in neural information processing systems 32: Annual conference on neural information processing systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada, pp 2120–2129
9. Shchur O, Bilos M, Günnemann S (2020) Intensity-free learning of temporal point processes. In: 8th international conference on learning representations, ICLR 2020, April 26-30, 2020, Addis Ababa, Ethiopia. OpenReview.net
10. Enguehard J, Busbridge D, Bozson A, Woodcock C, Hammerla N (2020) Neural temporal point processes for modelling electronic health records. In: Proceedings of the machine learning for health NeurIPS workshop, Proceedings of machine learning research, vol 136, pp 85–113. PMLR
11. Mei H, Wan T, Eisner J (2020) Noise-contrastive estimation for multivariate point processes. In: Advances in neural information processing systems, vol 33. Curran Associates Inc, pp 5204–5214
12. Sütfeld LR, Brieger F, Finger H, Füllhase S, Pipa G (2020) Adaptive blending units: trainable activation functions for deep neural networks. In: Intelligent computing - proceedings of the 2020 computing conference, volume 3, advances in intelligent systems and computing, vol 1230. Springer, pp 37–50
13. Rubanova Y, Chen TQ, Duvenaud D (2019) Latent ordinary differential equations for irregularly-sampled time series. In: Advances in neural information processing systems 32: annual conference on neural information processing systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC Canada, pp 5321–5331
14. Jia J, Benson AR (2019) Neural jump stochastic differential equations. In: Advances in neural information processing systems 32: annual conference on neural information processing systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC Canada, pp 9843–9854
15. Chen RTQ, Amos B, Nickel M (2021) Learning neural event functions for ordinary differential equations. In: 9th international conference on learning representations, ICLR 2021, virtual only, May 3–7, 2021
16. Chen RTQ, Amos B, Nickel M (2021) Neural spatio-temporal point processes. In: 9th international conference on learning representations, ICLR 2021, virtual only, May 3–7, 2021
17. Montavon G, Samek W, Müller KR (2018) Methods for interpreting and understanding deep neural networks. Digit Sig Process 73:1–15
18. Jacot A, Hongler C, Gabriel F (2018) Neural tangent kernel: Convergence and generalization in neural networks. In: Advances in neural information processing systems 31: annual conference on neural information processing systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada, pp 8580–8589
19. Chen M, Jiang H, Liao W, Zhao T (2019) Efficient approximation of deep relu networks for functions on low dimensional manifolds. In: Advances in neural information processing systems 32: annual conference on neural information processing systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada, pp 8172–8182
20. Xu K, Zhang M, Li J, Du SS, Kawarabayashi KI, Jegelka S (2021) How neural networks extrapolate: From feedforward to graph neural networks. In: 9th international conference on learning representations, ICLR 2021, virtual only, May 3–7, 2021
21. Nguyen T, Raghu M, Kornblith S (2021) Do wide and deep networks learn the same things? Uncovering how neural network representations vary with width and depth. In: 9th international conference on learning representations, ICLR 2021, virtual only, May, 3–7, 2021
22. Shokri R, Stronati M, Song C, Shmatikov V (2017) Membership inference attacks against machine learning models. In: 2017 IEEE symposium on security and privacy (SP), pp 3–18
23. Yeom S, Giacomelli I, Fredrikson M, Jha S (2018) Privacy risk in machine learning: Analyzing the connection to overfitting. In: 2018 IEEE 31st computer security foundations symposium (CSF), pp 268–282
24. Ramsauer H, Schäfl B, Lehner J, Seidl P, Widrich M, Gruber L, Holzleitner M, Adler T, Kreil D, Kopp MK, Klambauer G, Brandstetter J, Hochreiter S (2021) Hopfield networks is all you need. In: 9th international conference on learning representations, ICLR 2021, virtual only May 3–7, 2021
25. Mildenhall B, Srinivasan PP, Tancik M, Barron JT, Ramamoorthi R, Ng R (2020) NeRF: Representing scenes as neural radiance fields for view synthesis. In: Computer vision - ECCV 2020 - 16th european conference, Glasgow, UK, August 23-28, 2020, Proceedings, Part I, vol 12346, pp 405–421
26. Vaswani A, Shazeer N, Parmar N, Uszkoreit J, Jones L, Gomez AN, Kaiser L, Polosukhin I (2017) Attention is all you need. In: Advances in neural information processing systems 30: Annual conference on neural information processing systems 2017, December 4-9, 2017, Long Beach, CA USA, pp 5998–6008
27. Devlin J, Chang MW, Lee K, Toutanova K (2019) BERT: Pre-training of deep bidirectional transformers for language

understanding. In: Proceedings of the 2019 conference of the north american chapter of the association for computational linguistics: Human language technologies, vol 1 (Long and Short Papers), pp 4171–4186
28. Dosovitskiy A, Beyer L, Kolesnikov A, Weissenborn D, Zhai X, Unterthiner T, Dehghani M, Minderer M, Heigold G, Gelly S, Uszkoreit J, Houlsby N (2021) An image is worth 16x16 words: transformers for image recognition at scale. In: 9th international conference on learning representations, ICLR 2021, Virtual Only May 3–7, 2021
29. Sill J (1998) Monotonic networks. In: Advances in neural information processing systems, vol 10, pp 661–667
30. You S, Ding D, Canini KR, Pfeifer J, Gupta MR (2017) Deep lattice networks and partial monotonic functions. In: Advances in neural information processing systems 30: Annual conference on neural information processing systems 2017, December 4-9, 2017, Long Beach, CA, USA, pp 2981–2989
31. Wehenkel A, Louppe G (2019) Unconstrained monotonic neural networks. In: Advances in neural information processing systems 32: Annual conference on neural information processing systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada, pp 1543–1553
32. Zhuang T, Zhang Z, Huang Y, Zeng X, Shuang K, Li X (2020) Neuron-level structured pruning using polarization regularizer. Adv Neural Inf Process Syst, vol 33
33. Mescheder LM, Geiger A, Nowozin S (2018) Which training methods for gans do actually converge? In: Proceedings of the 35th international conference on machine learning, ICML 2018, Stockholmsmässan, July 10-15, 2018, Proceedings of machine learning research, vol 80, Stockholm, Sweden, pp 3478–3487. PMLR
34. Loshchilov I, Hutter F (2019) Decoupled weight decay regularization. In: 7th international conference on learning representations, ICLR 2019, May 6–9, 2019, New Orleans, LA, USA
35. Lobov SA, Mikhaylov AN, Shamshin M, Makarov VA, Kazantsev VB (2020) Spatial properties of stdp in a self-learning spiking neural network enable controlling a mobile robot. Front. Neurosci. 14:88
36. Naveros F, Luque NR, Ros E, Arleo A (2020) Vor adaptation on a humanoid icub robot using a spiking cerebellar model. IEEE Trans Cybern 50(11):4744–4757
37. Tang G, Shah A, Michmizos KP (2019) Spiking neural network on neuromorphic hardware for energy-efficient unidimensional slam. In: 2019 IEEE/RSJ international conference on intelligent robots and systems (IROS), pp 4176–4181
38. Mozafari M, Kheradpisheh SR, Masquelier T, Nowzari-Dalini A, Ganjtabesh M (2018) First-spike-based visual categorization using reward-modulated stdp. IEEE Trans Neural Netw Learn Syst 29(12):6178–6190
39. Zhu L, Dong S, Li J, Huang T, Tian Y (2020) Retina-like visual image reconstruction via spiking neural model. In: 2020 IEEE/CVF conference on computer vision and pattern recognition (CVPR), pp 1435–1443
40. Yu Q, Yan R, Tang H, Tan KC, Li H (2016) A spiking neural network system for robust sequence recognition. IEEE Trans Neural Netw Learn Syst 27:621–635
41. Song Z, Xiang S, Ren Z, Han G, Hao Y (2020) Spike sequence learning in a photonic spiking neural network consisting of vcsels-sa with supervised training. IEEE J Sel Top Quant Electron 26(5):1–9
42. Maciag PS, Kryszkiewicz M, Bembenik RL, Lobo J, Del Ser J (2021) Unsupervised anomaly detection in stream data with online evolving spiking neural networks. Neural Netw 139:118–139
43. Zhang M, Qu H, Belatreche A, Chen Y, Yi Z (2019) A highly effective and robust membrane potential-driven supervised learning method for spiking neurons. IEEE Trans Neural Netw Learn Syst 30(1):123–137
44. Wang X, Lin X, Dang X (2020) Supervised learning in spiking neural networks: A review of algorithms and evaluations. Neural Netw 125:258–280
45. Yang S, Gao T, Wang J, Deng B, Lansdell B, Linares-Barranco B (2021) Efficient spikedriven learning with dendritic event-based processing. Front Neurosci, vol 15. Publisher: Frontiers
46. Huh D, Sejnowski TJ (2018) Gradient descent for spiking neural networks. In: Advances in neural information processing systems 31: Annual conference on neural information processing systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada, pp 1440–1450
47. Yang S, Wang J, Zhang N, Deng B, Pang Y, Azghadi MR (2021) CerebelluMorphic: large-scale neuromorphic model and architecture for supervised motor learning. IEEE Trans Neural Netw Learn Syst, pp 1–15
48. Yang S, Deng B, Wang J, Liu C, Li H, Lin Q, Fietkiewicz C, Loparo KA (2019) Design of hidden-property-based variable universe fuzzy control for movement disorders and its efficient reconfigurable implementation. IEEE Trans Fuzzy Syst 27(2):304–318
49. Yang S, Deng B, Wang J, Li H, Lu M, Che Y, Wei X, Loparo KA (2020) Scalable digital neuromorphic architecture for large-scale biophysically meaningful neural network with multi-compartment neurons. IEEE Trans Neural Netw Learn Syst 31(1):148–162
50. Zhao Q, Erdogdu MA, He HY, Rajaraman A, Leskovec J (2015) SEISMIC: A self-exciting point process model for predicting tweet popularity. In: Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining, August 10-13, 2015. ACM, Sydney, NSW, Australia, pp 1513–1522

Sishun Liu is a full-time postgraduate student in the Department of Computer Science at Southwest University. His main research area is modeling irregular temporal-distributed sequences by temporal point process.

Li Li is a full professor in the Department of Computer Science at Southwest University. She obtained her Ph.D degree from Swinburne University of Technology, Melbourne, Australia. Her main research areas include machine learning, artificial intelligence in education. She has authored/co-authored over 120 papers in journals and conferences and 2 books. She is the member of IEEE and ACM.