University of Manchester
School of Computer Science
Project Report 2024

Event Sequence Prediction with Linear-Time Sequencing State Space Model

Author: Xinwei Niu

Supervisor: Dr. Mauricio Álvarez

Abstract

Event Sequence Prediction
with Linear-Time Sequencing State Space Model

Author: Xinwei Niu

Neural Temporal Point Processes(TPPs) are models specified for tasks on event sequence prediction, which are mostly based on Recurrent Neural Networks(RNNs) and transformer architecture in recent studies. Mamba architecture[8] is a novel deep learning architecture using the discretized State Space Models (SSM)[10]. As an attention-free architecture, Mamba shows great performance on sequence modelling and prediction tasks compared with transformer-based models. However, it is still uncertain about the performance of Mamba in solving complex time and event prediction tasks and handling event sequence datasets without the state-of-the-art attention mechanism[27].
In this paper, by modifying the Mamba architecture into Neural Temporal Point Process models, without using any attention mechanism, the Mamba Temporal Point Process (MambaTPP) shows better performance not only on time consumption but also prediction accuracy compared with LSTM-based and Transformer-based TPPs trained and evaluated on real-world event sequence datasets such as StackOverflow and Electronic Health Records (EHR).

Supervisor: Dr. Mauricio Álvarez

Acknowledgements

I would like to express my deepest gratitude to my supervisor, Dr. Mauricio Álvarez, for his indispensable assistance and feedback on this project.

Also, I would like to acknowledge the assistance given by Research IT and the use of the Computational Shared Facility at The University of Manchester.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Event sequences are sequence data that widely appear in our everyday life. For example, through the taxi data of carrying travellers around the city with pick-up and drop-off times[3] we can predict the patterns of travellers taking trips in a city. On the other hand, event sequence data such as earthquake records and weather records can be applied in the area of forecasting tasks. Similarly, by capturing data such as blogs and sharing on social media, as each action has corresponding timestamps it is simple to convert the action records into event sequences.

In recent studies on sequence prediction, event sequence prediction tasks have become a trend in the healthcare, delivery, and social media industries. Unlike time series data which only contains discrete time and time-dependent values, in the aspects of these industries the related data are in continuous time expressed with timestamps and categorized event types, such as admitting patient health records with the type of disease the patients suffer from with admission time[4] and blogs on social media with uploaded time and their relevant topics. Also, the event sequence data are asynchronous which shows the significance of the dynamics of time intervals. Therefore it is essential to have models that can process in continuous states instead of discrete for processing event sequences.

Point processes, which are models specified for the tasks on event sequence prediction are applied in various areas such as seismology, epidemiology, ecology, forestry, mining, hydrology, astronomy, ecology, and meteorology[24]. It involves different types of processes such as the ordinary Poisson point process which models event sequences in Poisson distribution, and self-exciting processes like the Hawkes process which models event sequences based on past events, which means the events that happened in the past are more likely to be happen again in the future time.

To increase the complexity of these traditional point processes and try to better model event sequence, Neural Temporal Point Processes (TPPs) are proposed as models for processing event sequence tasks and implemented with encoder-decoder models such as Recurrent Neural Networks (RNNs) and mainly Transformer architecture in the past works. On RNN-based neural TPPs, although variants like Long Short-Term Memory(LSTM)[**?**] and Gated Recurrent Units(GRU) are difficult to capture long-term dependency. However long-term dependency has been shown to play a critical role in modelling event sequences[35]. For example, asset returns are generally expected in the long term instead of short-term in the stock exchanges. Another example can be described as in healthcare patients who suffer from chronic diseases such as diabetes and cancer are unlikely to capture the symptoms at the beginning phase of the

diseases. Therefore TPPS need to capture long-term dependency. Also, RNNs are hard to train due to the gradient explosion and gradient vanishing[21].

Attention mechanism[27], which is the core property of Transformer architecture, is the state-of-the-art architecture in sequence modelling as it is better at modelling both short-term and long-term dependencies and has faster training time compared with RNNs. Also, Transformer architecture can be parallelized to improve the model's efficiency. However, the attention mechanism requires matrix multiplications on its Query, Key and Value matrices has shown its time-consuming drawback on inference as the time complexity of attention mechanisms which is quadratic, the time consumption of Transformer-based TPPs is enormous during predicting event sequence datasets with large sizes.

Mamba architecture is a novel deep learning architecture based on Structured State Space Models (SSM), a recent class of models used for sequence data. Unlike Transformer architecture, the time complexity of Mamba on inference and training are both linear. As a competitor to the popular state-of-the-art Transformer, Mamba, which even without the need for Multilayer Perception (MLP) blocks and attention, has shown its greater performance on language modelling, DNA modelling and audio generation tasks compared with other Transformer-based models[8]. Also, Mamba architecture has shown its ability to model long-term dependencies on previous works of SSMs.

Motivated by the potential of Mamba architecture in processing event sequence data and seeking implementations of neural TPPs, even without the need for encoder-decoder structure, I proposed MambaTPP, a TPP model based on Mamba architecture which is the first application of Structured SSM in the area of TPP research.

1.2 Aims

The key aims of this project are focusing on adapting Mamba architecture for event sequence prediction tasks to extend the application of Selective Structured SSM, demonstrating the effectiveness of Mamba architecture in the inference stage, accomplishing comparison with other state-of-the-art neural TPP models which are already proven as the considerable models on event sequence prediction tasks, on evaluation metrics such as accuracy on event type prediction and Negative Log Likelihood (NLL) and discovering insights on future TPP studies. Also implementing the intensity modelling methods in two ways to have a comparison on which implementation is more fitted to the Mamba architecture on event sequence modelling.

1.3 Project Structure

The structure of the report is described as follows. Chapter 1 is the general introduction of the project and the core of the project MambaTPP. Chapter 2 provides the background and the literature review on the point processes related literature and algorithms of the neural temporal point processes with the details of the Mamba architecture. Chapter 3 describes the methodologies of the MambaTPP model including the algorithms and the model pipeline showing how it processes data and methods to learn the parameters. In Chapter 4 the MambaTPP is compared with the baseline models with different types of neural TPPs under 5 different real-world benchmark datasets and experimenting with different combinations of hyperparameters on hyperparameter tuning. Chapter 5 gives the conclusion of the model performance and the discussion of the potential ways to improve in future neural TPP research.

Chapter 2

Background

2.1 Chapter overview

This chapter gives the required background knowledge of point processes and event sequences with the algorithms and functions related to the point processes.

2.2 Machine Learning

Machine learning is the area of Artificial Intelligence that uses statistical algorithms to predict the information of unknown data. Nowadays, machine learning is widely applied to daily life such as personalising messages based on the user view history on social media, and recommendation of products during online purchasing. Some traditional machine learning algorithms such as Linear Regression have shown great potential in sequence predictions such as time series forecasting however they still have much to improve as the low complexity of the models and getting better quality in processing real-world data. Therefore Deep Learning models are involved in performing more complex tasks and potentially predicting data with better quality.

2.3 Deep Learning

Deep Learning models are machine learning models that use Artificial Neural Networks(ANN) to predict instead of traditional statistical methods. These models are designed to learn the features from data by training on a dataset. They are constructed with layers of artificial neurons and contribute to the model's ability to recognize complex patterns. These neurons are first randomly initialized and use back-propagation by iterating through the data to update their weights to get the optimal result.

Deep Learning models have shown significant outcomes in the works of processing unstructured data such as images, audio, and text since back-propagation first applied[14] and recently different types of ANNs are proposed for different tasks, such as Convolutional Neural Networks(CNNs) on image processing and Recurrent Neural Networks(RNNs) on sequence data. In processing sequence data, various Deep Learning models with recurrent structures have been applied to sequence modelling tasks.

2.4 Sequence models

Sequence models are a subset of the Machine Learning models that specifically process sequence data which can address the dependencies of the data at different places in the sequences. Predominantly, the sequence models are applied in the area of Natural Language Processing, with recurrent neural networks[23], Long Short-Term Memory[7] and the gated recurrent unit[2] being popular. Later the attention mechanism and Transformer architecture were proposed which are revolutionary to the NLP tasks as the Transformer architecture improves the efficiency as it is parallelizable, and excels in long-term dependency capturing because of the self-attention mechanism to capture the similarity between tokens of the given sequences.

2.5 Traditional Point Processes and Temporal Point Process

Point Processes[24] is a collection of models that model a random collection of points in some space. In practical applications such as hazard forecasting, each point indicates the type/location of events and/or the observed time of events. Figure 2.1 shows an example of the point process data recorded for the earthquakes that occurred in Southern California including the magnitude of the earthquakes and relative time. Figure referenced from [24].

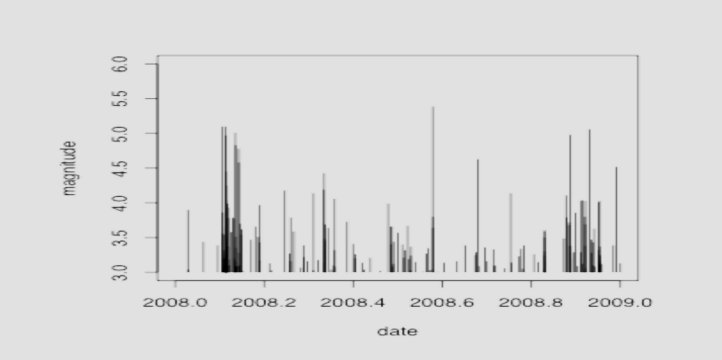

Figure 2.1: The point process data. [24]

The general way to define a point process N is, with data points being measured randomly on a separable metric space S with values in non-negative integers Z^+. The expression $N(A)$ defines the point process N models the data points lay inside the subset A of the space S. To specify temporal point processes, the data points are in the right-continuous space in the interval of $[0, T)$ and characterized into $\{t_0, t_1, ..., t_n\}$ with n inter-event times.

The most common and simple point process model is the Poisson point process which generalizes from the Poisson process, in which the data points follow the Poisson distribution and can be defined as with random variables $N(A_1), ..., N(Ak)$ which are independent Poisson random variables for subsets $A_1, ..., Ak$ in S, then N is a Poisson point process[24].

Focusing on temporal point processes, Hawkes processes[11] is the key concept as the self-exciting point processes models are popular in modelling temporal-clustered data points. Its conditional intensity $\lambda(t)$ can be defined as

$$\lambda(t) = \mu(t) + \sum_{t_i; t_i < T} v(t - t_i) \tag{2.1}$$

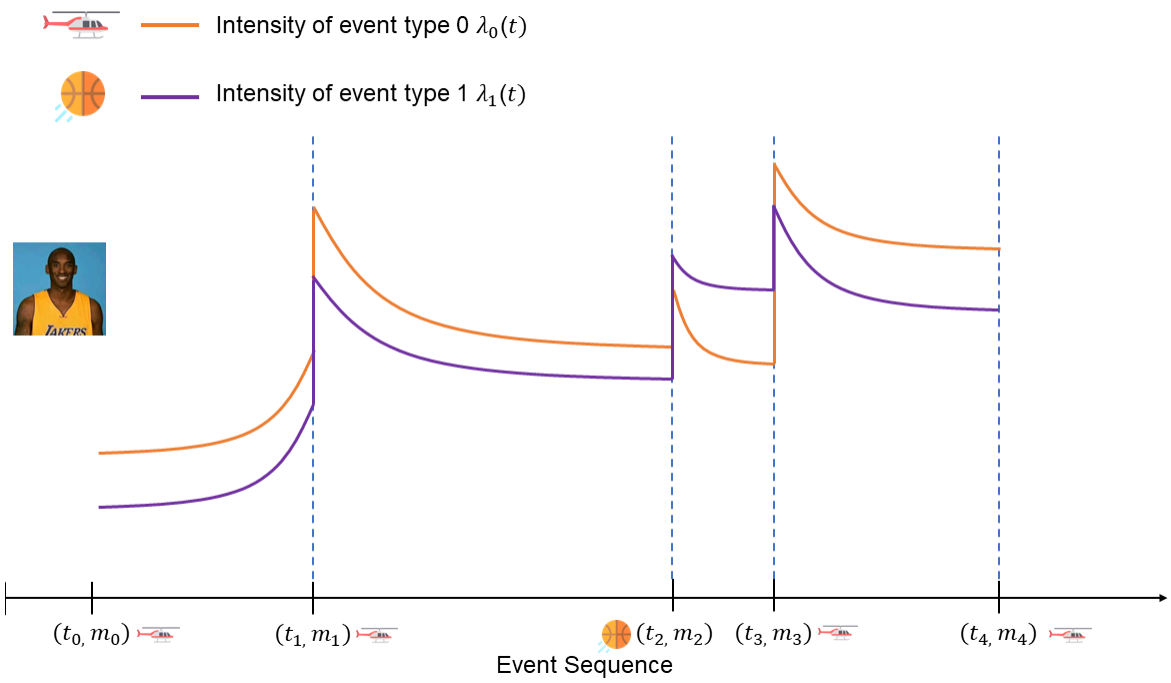

Figure 2.2: Demonstration of how event sequence data is being processed by neural TPPs.

where μ is the base intensity as a constant and $v(t-t_i)$ is the density of the cluster between t_i and t. As self-exciting point processes, the past events that occurred before are more likely to happen again, which means the future intensities of the clusters happen more are decay less compared with other clusters. Commonly in the works of point processes the density $v(\cdot,\cdot)$ is parameterized into $\alpha_{k,k'}(t) \cdot \kappa(t) \cdot \mathbb{1}_{t>0}$ where $\alpha_{k,k'}$ is the excitation parameter which models the base influence of the event type k' on the intensity of the event type k. The kick function K is the time-decaying influence which is an exponential function in most situations $\kappa = \exp(-\gamma t)$ where γ is the decaying parameter which models how much the intensity is decaying.

2.6 Neural Temporal Point Processes and Event Sequences

Event sequences are sequence data that each of the event sequences with size n of pairs of event type m_i and timestamp t_i that $S = \{(t_0, m_0), (t_1, m_1), ..., (t_n, m_n)\}$. In the sequence, event types are categorical marks that $m_i \in \mathcal{M} = \{0, 1, ..., C\}$ which C is the total number of events, and time are in the interval T that $\{\forall i < n; 0 \leq t_i \leq T$ and $t_{i-1} < t_i < t_{i+1}\}$.

Neural TPPs[25] is a class of sequencing models for event sequence prediction implemented with neural networks. For neural TPPs, the tasks are, by given part of the event sequence until to i-th pair, which can be denoted as history $H = \{(t_j, m_j); j < i\}$, to predict the i+1-th pair of time and event type. For neural TPP the event sequences are processed in the way shown in Figure 2.2. Generally, the procedures of neural TPPs are:

1. Embed the event types of the history $\mathcal{M}_H$ into vectors via the embedding layer with the size of C.

2. Embed the timestamps of the history into vectors.

3. Concatenate embedded vectors into the embedded history E_H.

4. Performing sampling on E_H to sample the conditional intensity $\lambda_k(t)$ of each event category k of m_i, which defined as

$$\lambda_k(t) = \lim_{\Delta t \downarrow 0} \frac{P(k \text{ in the interval } t_{i-1} \to t_{i-1} + \Delta t | E_H)}{\Delta t}$$

   which Δt is the inter-event time between m_{i-1} and m_i which can also be denoted as τ_{i-1}.

5. Calculate the conditional probability of the each event-time pair $P(m_i, t_i | E_H)$.

Figure 2.2 shows the demonstration of how event sequence data is being processed by neural TPPs. Neural TPPs predict future intensities, which are the probabilities of the events in the continuous time of each event type.

There are two types of neural TPPs which are autoregressive neural TPPs and continuous-time neural TPPs. Autoregressive neural TPPs using parametric distributions for obtaining the conditional probability density functions (PDFs) $f(\tau_i|\theta)$ and the survival functions $S(\tau_i|\theta)$ for the inter-event times, and then the intensities are derived from these terms with

$$\lambda_k(t) = \begin{cases} \frac{f(\tau_0|\theta)}{S(\tau_0|\theta)} & \text{if } t_0 < t < t_1 \\ \frac{f(\tau_1|\theta)}{S(\tau_1|\theta)} & \text{if } t_1 < t < t_2 \\ \vdots & \\ \frac{f(\tau_n|\theta)}{S(\tau_n|\theta)} & \text{if } t_n < t < T \end{cases} \tag{2.2}$$

The early works on neural TPPs are mostly on autoregressive neural TPPs, with processing on inter-event time using Recurrent Neural Networks and evaluation with discrete-time models with random point processes such as Poisson process and self-exciting point processes such as Hawkes Process[3]. With the appearance of attention mechanism and the Transformer architecture[27], as they showed their revolting performance on language modelling tasks, the research using modified transformer architecture and embedding event types and times by embedding layer and trigonometric positional embedding which are used on Transformer-based language models[35, 31].

Continuous-time neural TPPs differ from autoregressive neural TPPs as they do not require parametric distributions on sampling and differ from traditional point processes with no PDFs and survival functions involved. Continuous-time neural TPPs are initialized on latent states h which start at $t = 0$ and evolve in continuous time which has intervals of $(0, T]$. The latent states are then evolved and updated on each i-th occurrence:

$$h_{i-} = \text{Evolve}(h_{i-1}, t_{i-1}, t_i) \quad \text{in the time interval}$$
$$h_i = \text{Update}(h_{i-}, E_i) \quad \text{at the } i\text{-th occurrence}$$

The evolve function can evolve the latent state in the inter-event times and previous works on such models utilize ordinary differential equation (ODE) and exponential decay. The update function is similar to how RNN works to update the latent states. After getting the latent state at time t, the intensities of each event can be derived directly from the hidden states by using

non-linear functions such as Softplus and multi-layer perception(MLP) blocks, where $w_k^T h(t)$ is the linear projection layer.

$$\lambda_k(t) = \text{softplus}(w_k^T h(t))) \text{ for all categories } k \text{ at time t} \tag{2.3}$$

$$\text{or } \lambda_k(t) = \text{MLP}(h(t)) \text{ for all categories } k \text{ at time t} \tag{2.4}$$

The objective of training neural TPPs is the Negative Log Likelihood (NLL), which is the objective for both neural and traditional TPPs. This leads to using the Maximum Likelihood Estimation(MLE) to train the parameters by minimizing the NLL loss. The NLL of the input sequence S with n categories is:

$$-\log p(S) = -\sum_{i=1}^{n}\sum_{k=1}^{C} \mathbb{1}(m_i = k)\log\lambda_k(t_i) + \sum_{k=1}^{C}(\int_0^T \lambda_k(u)du) \tag{2.5}$$

which is generally inherited from the self-exciting Hawkes process where the proof can be found from [11]. The integral term of the NLL loss can also be expressed into the term into Λ as the cumulative intensity as it samples through the interval [0, T].

Between these two types of neural TPPs, the continuous-time neural TPPs make it easy to derive the intensities by only the current latent state and non-linear function. However, due to the need for integral estimation using the Monte Carlo method[19] on sampling the integral term in NLL losses, the cost of training time of continuous-time TPPs is higher than autoregressive neural TPPs which sample through predefined parametric distributions.

2.7 Related works on Neural Temporal Point Processes

The research on neural TPPs started from Recurrent Marked Temporal Point Processes(RMTPP)[3], using RNN to sample intensities that output the log-likelihood of event and time. In this work, comparing with traditional temporal point processes on synthetic Hawkes processes showed improved accuracy and Root Mean Squared Error(RMSE). RMTPP models the conditional intensities based on the original self-exciting intensity function as:

$$\lambda_k(t) = \exp({v^t}^\top \cdot h_j + w^t(t - t_j) + \mu^t) \tag{2.6}$$

where ${v^t}^\top$ is a column vector and w^t, μ^t are scalars. ${v^t}^\top \cdot h_j$, $w^t(t-t_j)$ and μ^t are the accumulated past influences from the embedded history, the inter-event influence and the base influence of the intensity[3]. This novel way to model the conditional intensity which later becomes the main intensity function of following neural TPPs models despite using different terms on the influences. To compute the integral term of the log-likelihood loss, Du et al. derived the intergral by the conditional density of the event sequences through the exponential of the accumulated intensities. Later Mei et al.[19] proposed Neural Hawkes Processes using the variant of RNN, which is the Long Short-Term Memory (LSTM)[7] and Monte Carlo sampling on integral estimation of log likelihood of next time and the first implementation of continuous-time neural TPPs. In this literature, the implementation uses discretization on a continuous-time LSTM. The Monte Carlo method on integral estimation of the continuous-time neural TPPs is generally followed as the estimation method based on this literature, which by given an interval [0, T], the model parameters, the history $H = \{(k_1, t_1)...\}$ and the target number of

Algorithm 1 Integral Estimation (Monte Carlo)[19]

Input: interval $[0,T]$; model parameters and events $(k_1,t_1),\ldots$ for determining $\lambda_j(t)$

$\Lambda \leftarrow 0$; $\nabla\Lambda \leftarrow \vec{0}$

for N samples **do**

  draw $t \sim (0,T)$ e.g., take $N > 0$ proportional to T

  for $j \leftarrow 1$ **to** K **do**

    Λ += $\lambda_j(t)$ via current model parameters

    $\nabla\Lambda$ += $\nabla\lambda_j(t)$ via back-propagation

  end for

end for

$\Lambda \leftarrow T\Lambda/N$; $\nabla\Lambda \leftarrow T\nabla\Lambda/N$ weight the samples

return $(\Lambda, \nabla\Lambda)$

times for sampling the intensity, the Monte Carlo can get the cumulative intensity by sampling through the interval to get Λ and its gradient $\nabla\Lambda$ where the details are shown in **Algorithm 1**[19]. The intensities are accumulated with intensities of all categories k at the sample times and after sampling the cumulative intensity and the gradient are weighted and get the final cumulative intensity. Also, Mei et al. found that by substituting the exponential function from the work of RMTPP by using Softplus function, which is the smoothed approximation of Rectified Linear Unit(ReLU) which constraints the output to be positive used on modelling intensities as the intensities are positive and avoid the intensities to be 0 which can cause infinitely training with bad log-likelihood losses and predicting event types and times based on bad log-likelihood on inference.

However, RNN-based models have the drawbacks of having weaknesses in modelling long-term dependencies in the sequences and problems of gradient vanishing or exploding. With the work of Vaswani et al.[27] on attention mechanism that can capture the similarities of the contexts through key, query and value, the Transformer architecture shows its advantages in modelling short-term and long-term dependencies and is easier to train compared with RNNs due to the problem of gradient explosion and gradient vanishing of RNNs[21]. Also, the Transformer architecture has shown great performance on Natural Language Processing(NLP) tasks with applications in Large Language Models and language translation tasks[27].

To discover the potential of the Transformer architecture process temporal information and event sequence in continuous time, The Self-Attentive Hawkes Process is proposed by Zhang et al.[32] as the first neural TPP using Transformer architecture. In this work, the positional embedding method is first tried on temporal encoding and takes inter-event time intervals into consideration. The temporal encoding pe is computed based on the positional encoding in the work of Transformer architecture[27] and the dimension of the historical hidden states D encoded into the same dimension with the embedded event types sequence tp. Given an event-time pair (m_i,t_i) the temporal embedding can be calculated as:

$$pe^{D}_{(m_i,t_i)} = \sin(\omega_k \times i + w_k \times t_i) \tag{2.7}$$

where i is the absolute position of the event-time pair and D is the target embedding dimension. ω_k is the unchanged predefined angle frequency and w_k is the scaling vector to scale the time t_i into k-dimensional vector.

The embedded history of the i-th event-time pair E_{H_i} is then computed by adding the embedded time sequence and the embedded event types sequences as:

$$E_{H_i} = tp_i + pe^k_{(m_i,t_i)} \tag{2.8}$$

With the embedded history from the beginning of the sequence to the i-th event-time pair, to compute the next intensity at t_{i+1} of the event type k, the hidden states can be therefore computed as:

$$\boldsymbol{h}_{u,i+1} = \left(\sum_{j=1}^{i} f\left(\boldsymbol{x}_{i+1}, \boldsymbol{x}_j\right) g\left(\boldsymbol{x}_j\right) \right) / \sum_{j=1}^{i} f\left(\boldsymbol{x}_{i+1}, \boldsymbol{x}_j\right) \tag{2.9}$$

where $\boldsymbol{x}_{i+1}$, $\boldsymbol{x}_j$ and $g_{\boldsymbol{x}_j}$ can be considered as the query, key and value of the self-attention mechanism. In this implementation $g(\cdot)$ is the non-linear activation function and $f(\cdot,\cdot)$ is the similarity function as an embedded Gaussian:

$$f(\boldsymbol{x}_{i+1}, \boldsymbol{x}_j) = \exp(\boldsymbol{x}_{i+1} \boldsymbol{x}_j^{\mathsf{T}}) \tag{2.10}$$

In this literature, the Self-Attentive Hawkes still evaluates the intensity based on the intensity function from equation 2.1. It uses three non-linear activation functions, which are GELU[12] for the base intensity μ and the starting intensity η, and Softplus on the decaying parameter γ.

The following other Transformer-based neural TPPs, the Transformer Hawkes Process by Zuo et al.[35] and Attentive Neural Hawkes Process by Yang et al.[30] are focused on the ways to embed history using sinusoidal embedding for time embedding and better way to assemble attention layers. In the work of the Transformer Hawkes Process, the events are embedded using one-hot encoding which embeds category marks into the vectors with the size of k and the elements are all zeros except the element at k-th index is one. Also, Zuo et al.[35] attempted to use the combination of RMSE on time and cross-entropy loss on event prediction instead of only NLL. Similar to Neural Hawkes Process and RMTPP, Transformer Hawkes Process models the intensity with history factors using "current influence" takes from the inter-event times $\Delta t = t_i - t_{i-1}$, "history influence" which samples through the model, and "base influence" represents probability of occurrence of events without considering any history information which is initialized using Xavier normal distribution[36, 6]. The intensities are also sampled using Softplus function like the previous models.

In the work by Yang et al.[30], the Attentive Neural Hawkes Process is based on the work of the Neural Hawkes Process which substitutes LSTM to multi-head attention and implements the history embedding method by concatenating the embedded vectors and the times used on sampling. In this work, the current influence, the history influence and the base influence are not even needed to model the conditional intensities which is the requisite term in most of the previous works, in this work the intensity is directly modelled from the output of the multi-head attention layers. Differing from the Transformer Hawkes Process and the Self-Attentive Hawkes Process, the temporal embedding used in the Attentive Neural Hawkes Process is based on the statistics of the training dataset and is explained and shown in Equations 3.1 and 3.2 in section 3.4. In general, the characteristics and the types of these state-of-the-art neural TPP are shown in Table 2.1.

With the works on RNN-based and Transformer-based neural TPPs, these neural TPPs are implemented into industries such as healthcare[4] to discover the pattern of the trajectory of sicknesses of patients using real-world electronic health records. Also, the TPPs are applied as recommender systems in purchasing by giving the customers' purchase history to give time-sensitive recommendations such as giving out promotions[25] .

Neural TPP Models			
Model	**Architecture**	**Type**	**Algorithms**
Recurrent Marked Temporal Point Processes[3]	RNN	Autoregressive	Logarithmic modelling, Intensity function
Neural Hawkes Processes[19]	LSTM	Continuous-Time	Intensity function
Self-Attentive Hawkes Processes[32]	Transformer	Continuous-Time	intensity function, first self-attention application
Transformer Hawkes Process[35]	Transformer	Continuous-Time	RMSE + Cross-Entropy, Intensity function
Attentive Neural Hawkes Process[30]	Transformer	Continuous-Time	No decay functions, Concatenating hidden vectors

Table 2.1: Comparison of Neural TPP Models

2.8 Structured State Space Models

In the area of mechanical engineering, State Space Models(SSMs) are widely used in control systems with applications in signal processing and implementation of Kalman filters. As a continuous signal modelling model, SSMs take 1-dimensional input $u(t)$ with a latent state $x(t)$ with the size of N for N-th order system and can be defined with two equations:

$$x'(t) = \boldsymbol{A}x(t) + \boldsymbol{B}u(t) \tag{2.11}$$

$$y(t) = \boldsymbol{C}x(t) + \boldsymbol{D}u(t) \tag{2.12}$$

where $\boldsymbol{A}$, $\boldsymbol{B}$, $\boldsymbol{C}$, $\boldsymbol{D}$ are learnable parameters and $y(t)$ is the projected 1-dimensional output.

As SSMs are widely used in signal modelling, the tries on modifying SSMs to sequence modelling models began as the past works have shown that original SSMs are working poorly on sequence modelling tasks such as using linear first-order ODEs to derive exponential functions, causing gradients scaling exponentially and problems of gradient exploding and vanishing[21] which are also the common problems of RNNs.

In the work on Structured State Space Model (S4) by Gu et al.[10], the parameter $\boldsymbol{D}$ has been omitted as the $\boldsymbol{D}u$ in Equation 2.12 can be considered as skip connection and easy to compute. The main object of this work is to discretize the state space model, where the input signal $u(t)$ is being discretized into $u(k\Delta)$, and also construct the learnable parameter $\boldsymbol{A}$ using HiPPO theory of continuous-time memorization[9] instead of random matrices to address long-term dependencies, which has shown the improved performance of SSMs on MNIST dataset benchmark by 38%, which can be determined as

$$\boldsymbol{A}_{nk} = \begin{cases} (2n+1)^{\frac{1}{2}}(2k+1)^{\frac{1}{2}} & \text{if } n > k \\ n+1 & \text{if } n = k \\ 0 & \text{if } n < k \end{cases} \tag{2.13}$$

Discretization is the method used on continuous inputs which converts the input data from continuous into discrete data or category marks. As the goal of the discretized SSM is to process sequence data such as tokenized texts represented in the form of token ids at different positions, the discretization can convert the required and output data from continuous functions into discrete data points. Such processes can be achieved by using Euler's method which takes the step size Δ and calculates the gradient to discretize. In the implementation of Mamba architecture, the discretization process is used with zero-order hold discretization instead which uses a rectangular function. Discretization and diagonalization are the most important parts of the S4 model. By discretization, the discretized SSMs can process the sequence $(p_1, p_2, ...)$ instead of getting the input as a function $u(t)$ and discretized with **step size** Δ based on bilinear method to convert the parameter $\boldsymbol{A}$ into an approximation $\bar{\boldsymbol{A}}$. Then the discretized SSM can be defined as

$$\begin{array}{ll} x_k = \overline{\boldsymbol{A}} x_{k-1} + \overline{\boldsymbol{B}} u_k & \overline{\boldsymbol{A}} = \exp(\Delta \boldsymbol{A}) \\ y_k = \overline{\boldsymbol{C}} x_k & \overline{\boldsymbol{B}} = (\Delta \boldsymbol{A})^{-1}(\exp(\Delta \boldsymbol{A}) - \boldsymbol{I}) \cdot \Delta B \quad \overline{\boldsymbol{C}} = \boldsymbol{C} \end{array} \tag{2.14}$$

After discretization, the SSMs can be applied as recurrent sequence modelling models because of processing input tokens recurrently and can be utilized as sequence modelling neural networks, where $x(t)$ can be considered as the hidden state equivalent with other sequence modelling networks. With two modes of the S4 models the model therefore can reach linear time training and inference by convolution and recurrent mode respectively.

Discretized SSMs can also be computed in the way of convolution, by computing the convolution kernel $\overline{\boldsymbol{K}} = (\boldsymbol{C}\overline{\boldsymbol{B}}, \boldsymbol{C}\overline{\boldsymbol{A}\boldsymbol{B}}, \ldots, \boldsymbol{C}\overline{\boldsymbol{A}}^{L-1}\overline{\boldsymbol{B}})$ where L is the length of the input sequence. The output y can then be calculated directly by performing convolution $y = u * \overline{\boldsymbol{K}}$.

The bottleneck of computing the discrete SSMs is they require multiple matrix multiplications of $\overline{\boldsymbol{A}}$ such as the convolution kernel computation which requires $O(N^2 L)$ operations and $O(NL)$ space. To overcome this bottleneck, all HiPPO matrices $\boldsymbol{A}$ have been diagonalized to achieve linear-time processing with Normal Plus Low-Rank which decompose $\boldsymbol{A}$ into a normal matrix and a low-rank matrix:

$$\boldsymbol{A} = V \Lambda V^* - P Q^\top = V \left(\Lambda - (V^* P)(V^* Q)^* \right) V^* \tag{2.15}$$

Where a unitary matrix $V \in \mathbb{C}^{N \times N}$, a diagonal matrix Λ[1] and low-rank factorization $P, Q \in \mathbb{R}^{N \times r}$. This therefore makes the performance of using S4 convolution mode with the time complexity of $O(N + L)$, where the proof can be found in [10].

One of the most influencing properties of the equations in equation 2.13 is that the model's dynamics are constant through time, which means all parameters, Δ,***A, B, C*** are fixed for all time steps. It is called linear time invariance and deeply affects the implementation of recurrence and convolutions. However, this property is discovered as the drawback of SSMs as linear time invariance models have limitations in modelling certain types of data.

Based on these basic properties of S4 and to overcome the limitation of linear time invariance, Mamba architecture introduced the selection mechanism using parametric projection to project input, which in Mamba is implemented with linear transformation on $\boldsymbol{B}$, $\boldsymbol{C}$ to make them input-dependent for recurrent SSMs which the structure is shown in Figure 2.3[8].

In Figure 2.3, the selection mechanism projects to parameters $\boldsymbol{B}_t$, Δ_t, and $\boldsymbol{C}_t$ to make them time-varying. Later Δ_t is used to discretize $\boldsymbol{B}_t$ and $\boldsymbol{A}$ for discretization. Figure referenced from [8].

[1] In this section Λ is not the cumulative intensity.

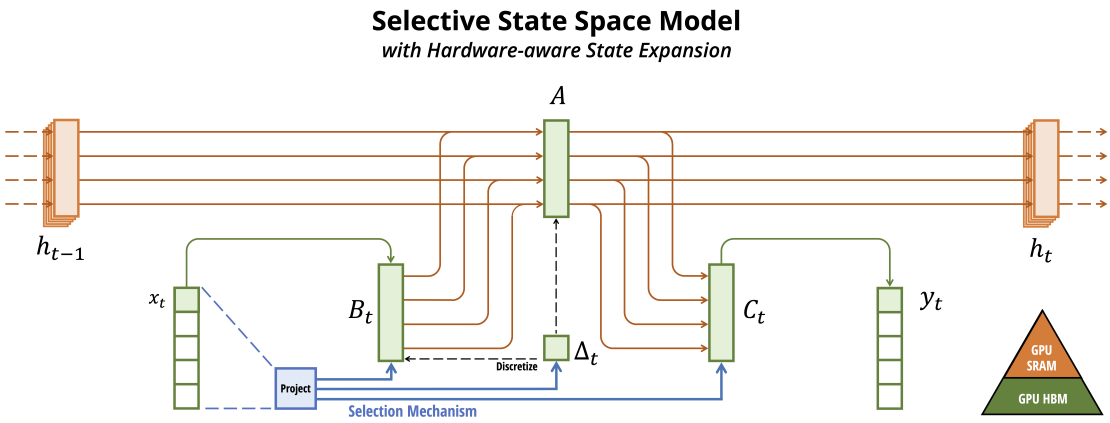

Figure 2.3: The structure of the Selective SSM layer.[8]

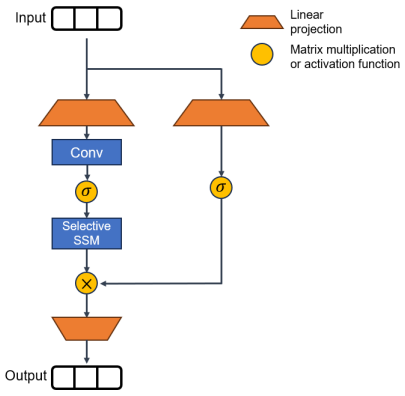

Figure 2.4: Structure of Mamba block

With selection mechanisms, the Mamba architecture can filter out noise tokens that are irrelevant to the data of interest. Also, it can reset the latent states at any time to remove extra history, therefore the performance should be improved monotonically related to context length theoretically.

Mamba architecture combines H3 architecture[5], which is the architecture that significantly improves the performance of the SSMs which makes them match up the performance of attention-based models, and the gated MLP block, which mainly has the structure of gated MLP with adding SSM layer into the main branch of the block with changing the first multiplicative gate into an activation function, which is selected with SiLU/Swish activation [22] for such architecture as the architecture of the Mamba Block is shown in Figure 2.4. Also, as Transformer-based models are privileged from parallelism which leading to high efficiencies. However the implementation on the original SSMs cannot be parallelized and the convolution mode of the Mamba architecture cannot be used for inference mode as it is time-variant, Therefore to achieve parallelism and high efficiency on both training and inference, the Mamba architecture uses parallel scan mechanism to achieve parallelism by using convolutional mode on training and recurrent mode for inference for efficiency. Likewise, the Mamba architecture uses kernel fusion, which reduces the numbers of memory IOs by fusing multiple operations into one operation, leading to time reduction on computations.

Previously the experiments comparing with Mamba-based models and other Transformer-based models and SSMs have shown its leading accuracy on computer vision[34], medical

image segmentations[18], DNA modelling tasks and audio modelling tasks[8] with faster time, which has shown the capability of the Mamba-based models in multiple areas of interests in Deep Learning.

Chapter 3

Methodology

3.1 Chapter Overview

This chapter gives the methodology of how MambaTPP is implemented based on, the information on the format of the data and the procedures of how MambaTPP processes the event sequence data and gives predictions.

3.2 Model Pipeline

To ensure the performance of the novel sequence modelling model that has shown great performance on long-term dependencies modelling tasks and Natural Language Processing tasks, the Mamba Temporal Point Process(MambaTPP) is proposed as the first neural TPP model that is implemented based on the Mamba architecture with Selective SSM layers. The MambaTPP is a continuous-time neural TPP that evaluates event sequences in continuous time. In the following explanations, suppose having an event sequence data $S = \{(t_0, m_0), (t_1, m_1), ..., (t_n, m_n)\}$ with the length of n, and for each event type m are categorized in $K = \{0, 1, 2, ..., k-1\}$ with k numbers of categories.

MambaTPP first processes the data into two separate sequences, $\mathcal{M}$ for only event types and $\mathcal{T}$ for only time. These data are loaded using dataloaders and with a collating function for padding and getting the masks on sequences. Later $\mathcal{M}$ is embedded with an embedding layer into $E_{\mathcal{M}}$, and $\mathcal{T}$ is embedded into $E_{\mathcal{T}}$ using sinusoidal embedding, where the detail is shown in Section 3.2.

Sample times t' are selected with evenly divided intervals in the inter-event times which is $\Delta\mathcal{T} = \mathcal{T}[:, 1:] - mathcalT[:, :-1]$ for estimating next time and integral estimation for NLL on times, and also being embedded into $E_{t'}$ using sinusoidal embedding. The embedded data $E_{\mathcal{M}}$ and $E_{\mathcal{T}}$ are then concatenated on the last dimension and sent into Mamba layers with the layer numbers of l. The output from the Mamba layers is projected and the Softplus activation function is to obtain the conditional intensity of each category and calculate the NLL loss based on summing up these conditional intensities and sampled integral term. In testing and validation, the thinning algorithm is used to sample the intensities in the sample times with the criteria of accepting and rejecting times to get the relative intensities at accept times to get the predictions on event types and event times.

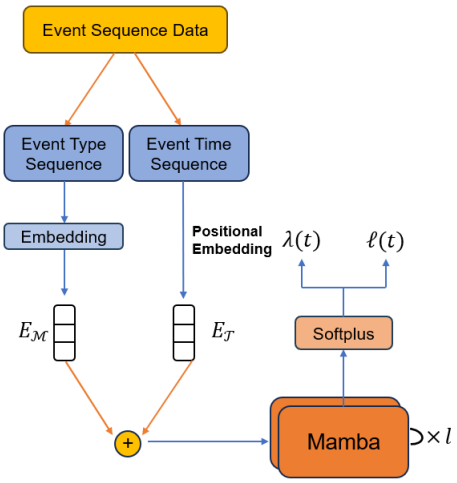

Figure 3.1: Model Pipeline

3.3 Event Sequence Data

The event sequence data for the MambaTPP are formatted into a dictionary with two elements with the keys of *dim_process* and *sequence*. The value of *dim_process* stores the number of categories of the whole dataset, and the value of *sequence* stores three sequences, which are $\mathcal{M}$, $\mathcal{T}$ and $\Delta\mathcal{T}$ which is the inter-time interval $\Delta\mathcal{T} = \{t_1, t_2 ...\} - \{t_0, t_1 ...\}$. Also, the number of categories of the training data is recorded for the target dimension of linear projections and the padding value for embedding. In the example shown in Figure 3.2, the *index_event* is the index of the event i, *type_event* is the type of the i-th event k_i expressed as an integer, *time_since_start* is the time of the i-th event t_i and *time_since_last_event* is the inter-event time of $t_i - t_{i-1}$.

The implementation of dataloaders for the event sequences is done by loading the 3 sequences first. As padding is needed on batch learning to pad the sequences to the length of the longest sequence in the current batch, a collating function is implemented, which is, first padding up all event type, event time and inter-event time sequences using the right padding where the padding value is the numbers of the categorical marks, and the padding and non-padding masks are computed for mask out the useful elements and padding elements in the later evaluations.

```
{'dim_process': 75,
{'idx_event': 36, 'type_event': 3, 'time_since_start': 5.9975, 'time_since_last_event': 0.2586111111111107}]]
```

Figure 3.2: The format of the preprocessed event sequence data.

3.4 Event Sequence Embedding

The way of how MambaTPP embeds event sequence data is by using an embedding layer for event types and sinusoidal embedding for event times. I trained the embedding layer by using the EMBEDDING() function in Pytorch to get the embedded event type sequences $\boldsymbol{W}_k^{D_m}$ with

the shape of $k \cdot D_m$, where D_m is the target embed size for event sequence encoding. Although the vanilla Mamba architecture does not use positional embedding as it processes discrete data that do not involve unequal time intervals, for processing event sequence data which is in continuous space, the time embedding is essential to model the intensities, the NLL loss, and the prediction of next event time.

Therefore on embedding methods to embed time sequence, as Zuo et al. [35] and Yang et al. [30] showed that using trigonometric functions improved the awareness of timestamp-like data, I adopt the sinusoidal embedding from Yang et al. [30] based on the positional embedding method by the Transformer architecture[27], which is defined as

$$E_{\mathcal{T}_d} = \sin\left(\frac{\mathcal{T}_d}{m \cdot (\frac{5M}{m})^{\frac{d}{D_\mathcal{T}}}}\right) \text{ if } d \text{ is even} \tag{3.1}$$

$$E_{\mathcal{T}_d} = \cos\left(\frac{\mathcal{T}_d}{m \cdot (\frac{5M}{m})^{\frac{d-1}{D_\mathcal{T}}}}\right) \text{ if } d \text{ is odd} \tag{3.2}$$

where d is the index of the timestamps in $\mathcal{T}$ and $D_\mathcal{T}$ is the target embed size of time sequence encoding. When the index of the current timestamp is odd, the embedded value of this timestamp is calculated by minus 1 on d and embedded using the cosine function. If the index of the current timestamp is even, the index d id directly used and embedded using the sine function.

On selection of constants m and M, as Yang et al[30] using m and M to increase the robustness of the model to fit different datasets as the time intervals would be much different, therefore m and M is decided based on the properties of the training data. For m, the shortest inter-event time from the whole training data is chosen, and M should be greater than all T in the training data which are

$$m = \min_{S \in \mathcal{D}_{\text{Train}}} \min_{s,s' \in \mathcal{T}_S} |s - s'| \tag{3.3}$$

$$M > \max_{T_S; S \in \mathcal{D}_{\text{Train}}} T_S \tag{3.4}$$

After getting the time sequence and the sample times, I use the inter-event times $\Delta\mathcal{T}$ as the input of the time by subtracting the sampling times from the original times. Then $E_\mathcal{M}$ and $E_{\Delta\mathcal{T}}$ are concatenated into E_H on the last dimension with the shape of $L \cdot (D_m + D_\mathcal{T})$.

3.5 Processing with Mamba Layers

Mamba layers are the most important and the basic components in MambaTPP. After the embedding procedure to get the embedded history E_H, E_H is then sent into the Mamba layers. As E_H is concatenated with sample times therefore the shape is $(B, L, D_m + D_\mathcal{T})$, where B is the batch size and L is the original sequence length. Then E_H is first normalized using Root Mean Square Layer Normalization, which is also known as RMSNorm by scaling the input into a learnable tensor, and then linear projected into x with the hidden dimension with the size of inner size T with a linear layer and passes into the Mamba layers sequentially. In each layer, x is passed through a 1-dimensional convolution operation and activated with SiLU/Swish activation function[22]. The required parameters of ***B, C*** and the discretized step size Δ are projected

Algorithm 2 The pseudocode for processing embedded history in MambaTPP

Require: $E_H : (B, L, D_m + D_{\mathcal{T}})$, number of layers $l > 0$
  initialize $\boldsymbol{A}$, Δ
  for *layer* $\leftarrow 0$ **to** l **do**
    $x : (B, L, T) \leftarrow \text{Linear}^{T}_{D_m + D_{\mathcal{T}}}(E_H)$
    $x : (B, L, T) \leftarrow \text{SiLU}(\text{Conv1d}(x))$
    $\boldsymbol{B} : (B, L, N) \leftarrow \text{Linear}^{N}_{T}(x)$
    $\boldsymbol{C} : (B, L, N) \leftarrow \text{Linear}^{N}_{T}(x)$
    $\Delta : (B, L, T) \leftarrow \text{Softplus}(\text{Broadcast}^{T}_{1}(\text{Linear}^{1}_{T}(x)) + \text{Parameter}_{\Delta})$
    $\overline{\boldsymbol{A}} : (B, L, D, N) \leftarrow \exp(\Delta \boldsymbol{A})$
    $\overline{\boldsymbol{B}} : (B, L, D, N) \leftarrow (\Delta \boldsymbol{A})^{-1}(\exp(\Delta \boldsymbol{A}) - \boldsymbol{I}) \cdot \Delta \boldsymbol{B}$
    $y : (B, L, T) \leftarrow \text{SSM}(\overline{A}, \overline{B}, C)(x)$
    $y : (B, L, D_m + D_{\mathcal{T}}) \leftarrow \text{Linear}^{D_m + D_{\mathcal{T}}}_{T}(y)$
    $x \leftarrow y$
  end for
  $y : (B, L, K) \leftarrow \text{Linear}^{K}_{D_m + D_{\mathcal{T}}}(y)$
  return y

from x because of the selection mechanism of the Mamba architecture to make them input-dependent, and $\boldsymbol{A}$ is initialized based on equation (2.13) to improve long-term dependencies by the selection mechanism. Next, the discretized parameters $\overline{\boldsymbol{A}}, \overline{\boldsymbol{B}}$ are processed based on the equation (2.14).

With all required parameters processed, x is then processed by the selective discrete-time SSM component with the discretized parameters processed above, gets the linear projected output y which has the same shape on x, and then x is assigned with the value of t and iterates through all the Mamba layers and finally normalized with RMSNorm to get the final output y.

To get the next intensity predictions at the next timestamp, the final output y is linear projected to the dimension of K and activated using Softplus for calculating the intensities of each category k at the time t. The details of how MambaTPP processes embedded history are shown in **Algorithm 2** which is generally using the procedures from the Mamba architecture.

3.6 Log-likelihood Estimation

After the MambaTPP model processes the embedding history E_H, it starts to estimate the log-likelihood by using **Algorithm 1** and Equation 2.5. The event sequence loss, which is the first term of the equation, can be easily computed after obtaining the intensities of the next time from the model. To calculate that, the type masks processed in dataloaders are used to check whether the predicted event types are equal to the ground truth. For the integral term, I followed the Monte Carlo method in **Algorithm 1** where the size interval is based on the inter-event time sequence, and samples along in this interval using linspace() function which evenly spaced the time interval. As it is an estimation method, the more samples are in this interval, the more accurate the estimation of this integral term will be. However, increasing the number of samples means that the time cost of log-likelihood loss approximation will be increased therefore it is crucial to choose the number of samples inside this interval. In the works of state-of-the-art implementations,

3.7 Continuous-Time Intensity Modelling

In the intensity modelling phase, as an implementation of a continuous-time neural TPP, the conditional intensity is sampled through a non-linear activation function. For MambaTPP, I followed two ways of sampling intensities, the first is based on the work of the Neural Hawkes Process[19] where the intensities are the combination of current, history and base intensity activated using the Softplus function. The output $y(t|E_H)$ is the historical influence of the intensities on each category mark k. The base influence is a constant being randomly initialized and the current influence is the multiplication of a random initialized matrix and inter-event time sequence which can be expressed as:

$$\lambda_k(t|E_H) : (B,L,1) = \text{Softplus}(\alpha(t_i - t_{i-1}) + \text{Linear}^K_{D_m}(y(t|E_H))[:,:,k] + \mu) \text{ for } k \text{ in } K \quad (3.5)$$

which the inter-event time sequence $\Delta\mathcal{T} = \{t_1 - t_0, t_2 - t_1, ...\}$ is already preprocessed in Section 3.3.

In another way of modelling intensity, I followed up with the implementation of AttNHP[30] by letting $\lambda_k(t|E_H)$ directly be the conditional intensity of category k without any other influences mentioned in the method above, which samples from time t with the embedded history E_H, and y is the output from the Mamba layers from Section 3.3. The conditional intensity of the category k is activated using Softplus, as it scales the output from the Mamba layers y into non-negative real numbers.

$$\lambda_k(t|E_H) : (B,L,1) = \log(1 + \exp(y_k)) \text{ for } k \text{ in } K \quad (3.6)$$

On the time prediction, the original thinning algorithm[17] used on traditional and autoregressive neural TPPs can also be applied for predicting next time t. Which is, first obtaining the upper bound of intensities of the i-th event type and event time, on all event types k at each sample time t that $\{\lambda^i_k(t); t \in [t_0, \infty)\}$ (however in the application it is impossible to sample with this wide interval therefore the interval is set into a reasonable positive number). Given these sample times, the upper bounds are derived by calculating the intensities at these times by MambaTPP and computing the maximum output intensities on each event type. The intensities are then sampled with the exponential distribution of the upper bound to act as the step on sampling time, which is, the sample time t_s keeps being incremented by the step until the accepted times are found. By multiplying a uniform distribution u with the interval (0, 1) to the upper bounds, the uniformly distributed upper bounds must lay inside the intensity at time t_s else being rejected indicating that the event is unlikely to occur at this time. Then the accepted times t^k_i of every k are recorded and the predicted event type $\hat{k}_i$ and the predicted event time $\hat{t}_i$ are chosen from the earliest accepted times $\hat{k}_i = \text{argmin} t^k_i$ and $\hat{t}_i = \min t^k_i$. In this practice, the distributions are implemented with uniform_() function and exponential_() function.

To match up with the state-of-the-art models that have better thinning methods, I adopted the way of Yang et al.[30] on sampling and predicting next time in continuous time which is, $\lambda_k(t)$ is a continuous function of the temporal embedding E_t, as E_H is in the compact space of $[-1,1]^d$ which is caused by the sinusoidal embedding, therefore $\lambda_e(t)$ is bounded. Then the numerical bound can be computed with any continuous function $f : \mathbb{R} \to \mathbb{R}$ to get the bounded interval of $\lambda_k(t)$ for all $t \in [t_0, \infty)$.

By predicting the time of the i-th event with this adaptive thinning method, the event time t_i has density $p_i(t) = \lambda(t)\exp(-\int_{t_{i-1}}^{t} \lambda(t')dt')$. With this approach to sample $\hat{t}$ the time can be sampled by the integral equation $\int_{t_i-1}^{\infty} t\, p_i(t)dt$ in the density $p_i(t)$ drawn with the thinning method.

For event-type prediction, the predicted event type is $\hat{k}$ which has the highest intensity among all K at the time t, which

$$\hat{k} = \text{argmax}_{k \in K} \lambda_k(t|E_H) \tag{3.7}$$

The predicted event types $\hat{k}$ and time t are used to evaluate the model performance based on the metrics in Section 3.6.

3.8 Training and Evaluation Metrics

Like all other traditional TPPs and neural TPPs, the MambaTPP is also trained based on NLL loss as mentioned in equation 2.5 which the derivation can be found in the previous works of TPPs[11, 19]. To model the NLL loss on both events and non-events, the Monte Carlo estimation method[19] is used on sampling times t to estimate the non-event NLL loss as the integral term in the equation 2.5 stands for the negated non-event log-likelihood loss, which match the number of samples to the number of the observed events to sample times t at training time as a fast approximation choice. On getting the event log-likelihood loss, by simply computing the term $-\sum_{i=1}^{n}\sum_{k=1}^{C}\mathbb{1}(m_i = k)\log\lambda_k(t_i)$ using the intensity $\lambda_k(t_i)$ computed from Equation 3.5 or Equation 3.6 and sum up for each category.

In the model evaluation phase, I have chosen 3 metrics that are deeply related to the performance of the neural TPP models, which are the event type prediction accuracy, RMSE of the event time, and the NLL loss. The accuracy and RMSE are evaluated by next-token prediction instead of the performance of the prediction on the last token of the sequence to get the general performance of the prediction ability of neural TPPs. Also, the next predicted event types are chosen using Minimum Bayes Risk(MBR) principles to follow the previous neural TPP works, which predict the event types under the intensities at the same timestamp and find the types that have the maximum intensity which is described in Section 3.7.

Chapter 4

Experiment

4.1 Chapter Overview

This chapter gives the experimental setup of the MambaTPP with the details of datasets used for benchmarking and the competitors. The metrics of the evaluations on real-world datasets are recorded with comparisons on other state-of-the-art neural TPP models and the results and combinations of hyperparameter tuning on MambaTPP.

4.2 Datasets

For experiments, to establish better comparisons with previous neural TPPs, I adopted five real-world benchmark datasets that are organized in the open Neural TPP benchmark framework Easy Temporal Point Processes constructed by Xue et al.[29] where the Taxi dataset the preprocessed version from RMTPP by Du et al.[3] and datasets preprocessed from Mei et al. [19] to facilitate comparison with previous works on neural TPPs, which are:

- **Taxi[28]**: A dataset which contains the taxi drop-off/pick-up data among five different boroughs operating in New York City, therefore there are $K = 10$ event types with the timestamp on each incident.
- **StackOverflow[16]**: A dataset contains the two-year records of users obtaining badges on a question-answering website with a total of $K = 22$ types of badges and awarded time.
- **MIMIC-II[15]**: A dataset preprocessed from electronic health records that contain the admission records of patients and have a total of $K = 75$ event types which are the recorded diseases and the admission time.
- **Amazon[20]**: A dataset contains the user reviews from January 2008 to October 2018. Each sequence is a user review of different types of products with $K = 16$ types and review time.
- **Retweet[33]**: A dataset includes the user retweeted history and categorized users into "small", "medium" and "large" with users having fewer than 120, fewer than 1363, and more than 1363 followers respectively.

Dataset	K	Number of Tokens			Sequence Length		
		Train	Validation	Test	Min	Mean	Max
Retweet	3	369000	62000	61000	10	41	97
MIMIC-II	75	1930	252	237	2	4	33
Amazon	16	288000	12000	30000	14	44	94
Taxi	10	51000	7000	14000	36	37	38
StackOverflow	22	90000	25000	26000	41	65	101

Table 4.1: Statistics of Selected Datasets

4.3 Compared Models

For comparison of the performance of MambaTPP with other state-of-the-art neural TPPs, I have chosen two other continuous-time neural TPPs based on Transformer architecture and one RNN-based neural TPP, which are

- **Transformer Hawkes Process (THP)[35]**: A continuous-time neural TPP based on Transformer and use trigonometric functions on event time embedding instead of original positional embedding by Vaswani et al.[27]. Unlike other neural TPPs that use NLL as the loss function, THP uses the sum of RMSE and Cross-Entropy loss for the loss function.
- **Attentive Neural Hawkes Process (AttNHP)[30]**: A continuous-time Transformer-based neural TPP based on Zuo et al.[35]. This work enhances the trigonometric embedding depending on the training dataset. Also for the AttNHP, the intensities are modelled directly through the hidden states of the last layer without any current, base or history influence.
- **Recurrent Marked Temporal Point Process (RMTPP)**[3]: The first autoregressive neural TPP implemented by RNN which evaluates event sequence data recurrently for the next event and time.

All of these baseline models except AttNHP are used in the Pytorch version from the EasyTemporalPointProcesses framework on neural TPPs[29] where all these models are based on original implementations and friendly on executing experiments under the same configurations. On AttNHP, as the implementation of the model in the framework performs poorly which has a huge difference compared with the performance of the original implementation, I used the original implementation of this model instead to give a fairer comparison.

4.4 Training Details and Hyperparameters

To perform comparisons as accurately as possible, and compared with Transformer-based TPPs the MambaTPP takes more parameters in the same numbers of the target dimension of the event type, the target dimension of the temporal embedding, and the number of layers, I control the number of parameters of all models to be almost the same **except** RMTPP. For neural TPPs, the main hyperparameters are the hidden dimension of the embedding D_m and the number of the layers l which is on the attention-based neural TPPs. For MambaTPP, the model also requires

the number therefore two hyperparameters needed to be tuned. To find the best set of hyperparameters, I trained my model with each combination of the embedded dimension of the event types sequence D_m and the number of layers l. I chose the candidates of $D_m \in \{8, 16, 32, 64\}$ and $l \in \{1, 2, 3\}$ with grid searching which tries out each of the potential combinations of hyperparameters. The hyperparameters are tuned automatically by Weighted and Biases(WandB)[1] with the *sweep* functionality by logging and integrating the log-likelihood loss, the accuracy and the RMSE information of the model to compare the model performance under different hyperparameter settings. These hyperparameter settings are applied on the StackOverflow and MIMIC-II datasets as they are featured in evaluating the performances of how the model addresses long-term and short-term dependencies respectively. In this phase, the number of sampling times is set to 10 times the number of events in the training set. After evaluating hyperparameters the results show that the combinations of $D_m \in \{16, 32\}$ and $l \in \{1, 2, 3\}$ can give optimal results on getting low enough validation NLL losses with both these two datasets where the results are shown in Appendix A.1.

To train the model, I used Adam[13] as the optimizer with the learning rate of 1×10^{-3}. On RMTPP, THP, and MambaTPP, the hyperparameters are the target event type embedding dimension D_m, the target time embedding dimension $D_{\mathcal{T}}$ and the number of layers l. As THP and AttNHP use multi-head attention therefore the number of heads is set to $h = 2$. The chosen hyperparameters for the main experiments are given in Table 4.2. The batch size for all the training datasets is set to 32, and training for 50 epochs for all TPP models to get the models with the highest log-likelihood loss in the validation phase. To get reliable results of how these models are performing and the robustness of the models, k-fold cross-validation is essential to get better comparisons. K-fold cross validation is a technique of dividing the dataset $\mathcal{D}$ into k sub-datasets that $\mathcal{D}_1, \mathcal{D}_2, ..., \mathcal{D}_k \in \mathcal{D}$ and $\{\forall i, j \leq k; D_i \cap D_k = \varnothing\}$, where k is an positive integer $k \in \mathbb{Z}^+$. For this experiment I used 5-fold cross-validation on all models with all datasets to get the average accuracy, RMSE and test losses to gain the general performance comparison and robustness analysis. For MambaTPP the model is evaluated with two implementations, one with the original thinning algorithm with traditional intensity modelling using Equation 2.1, and one with the direct intensity modelling from the hidden states based on equation 3.5 and the adaptive thinning method with the implementation on AttNHP[30]. The parameters of Monte Carlo estimation on all models are set with the number of sampling inter-event times of 20. Also, the early stopping technique is implemented, which is, if the model has already reached the lowest validation loss and the model starts to overfit, the training session will be terminated and the best model is saved to get the optimal performance and saves time on useless training with zero model improvements.

Computation Cost. All the experiments were conducted on the Computational Shared Facility(CSF3), the High Performance Computing cluster of the University of Manchester with a GPU node with 192GB RAM, two 16 logical cores CPUs (Intel(R) Xeon Gold 6130 @2.10GHz), and one NVIDIA Tesla V100 GPU for acceleration. For all models, the time to train all the datasets is about 1 hour except AttNHP is extremely time-consuming and takes about twice, even three times of the time consumption on the long-sequence datasets.

4.5 Evaluation and Statistics

The metrics of evaluating the models are the log-likelihood loss by negating the NLL loss used for minimizing during training, the event prediction and RMSE on the inter-event time

Hyperparameter Settings		
Model	**Hyperparameter**	**Value**
RMTPP	*hidden_size*	32
	time_emb	16
	layers	2
THP	*hidden_size*	64
	layers	2
	heads	2
AttNHP	*hidden_size*	32
	time_emb	16
	layers	1
	heads	2
MambaTPP	*hidden_size*	16
	time_emb	16
	layers	2

Table 4.2: Hyperparameter settings on neural TPP models.

mentioned in Section 3.6. The log-likelihood is the main objective for the neural TPPs to maximize through MLE which is learned by minimizing the NLL losses and can be computed as:

$$\log p(S) = \sum_{i=1}^{n} \sum_{k=1}^{C} \mathbb{1}(m_i = k) \log \lambda_k(t_i) - \sum_{k=1}^{C} \left(\int_0^T \lambda_k(u) du \right) \tag{4.1}$$

by simply negating every term in Equation 2.5. The accuracy is calculated by

$$accuracy = \frac{\sum_{i=1}^{L} \mathbb{1}(\hat{k}_i = k_i)}{L} \tag{4.2}$$

where sums up the true positives by matching up whether the predicted event type $\hat{k}$ is equal to the ground truth k divided by the total number of events.

The RMSE is calculated by the predicted inter-event times from sampling and the ground truth of the inter-event times by:

$$RMSE = \sqrt{\frac{\sum_{i=1}^{L} (\Delta t_i - \Delta \hat{t}_i)^2}{L}} \tag{4.3}$$

which calculates the square root of the mean squared difference between the predicted inter-event time $\Delta \hat{t}$ and the ground truth Δt.

In the comparison, all models are evaluated based on log-likelihood loss, accuracy and RMSE except AttNHP is evaluated without RMSE which because of the implementation by Yang et al.[30] only accept predicts the next time when the batch size is 1. Both accuracy and log-likelihood loss are the larger the better, and RMSE is the smaller the better.

4.6 Result Analysis

After fitting MambaTPP and other state-of-the-art TPPs to the benchmark datasets, the models are evaluated on the evaluation metrics with results showing from Figure 4.1 to Figure 4.5 on the **Amazon, Taxi, Retweet, StackOverflow** and **MIMIC-II** datasets. Also, Table 4.3 to Table 4.5 shows the overall performance of all models on all datasets. N.A. in the table stands for Not Available as RMSE not available mentioned in Section 4.4. The error bars in the figures show the standard deviation of the metrics after five runs of each model on every dataset. The best metrics are underlined in Table 4.3 to Table 4.5.

As shown in the results, the results demonstrate that MambaTPP not only aligns closely with all state-of-the-art TPPs but also outperforms them on some of the tasks. Looking at Table 4.3, MambaTPP has the best accuracy across StackOverflow, Taxi, and Amazon datasets with accuracy of 45.18%, 91.59% and 35.46% which outperforms all other models a little. MambaTPP's performance is marked by considerable accuracy across all datasets, coupled with low standard deviations in the accuracy of 0.0012, 0.0072, 0.0006, 0.0225 and 0.0082 respectively, which underscores the robustness of MambaTPP in predicting event types on both long and short sequences. In terms of RMSE evaluation results shown in Table 4.5, MambaTPP consistently shares the lowest RMSE and the RMSE standard deviations with THP except for the StackOverflow dataset. However, it's noteworthy that MambaTPP exhibits reasonable standard deviations with an average of 0.0001 on short-sequence datasets(MIMIC-II and Taxi) and 0.0432 on long-sequence datasets(Amazon, StackOverflow and Retweet), further attesting to its reliability and robust performance on the time prediction tasks. On log-likelihood loss results which are shown in Table 4.4, we can see that that **AttNHP** has the highest log-likelihood losses in 3 out of 5 datasets, and MambaTPP has the highest in evaluating the StackOverflow dataset. MambaTPP also has the second highest log-likelihood losses in Amazon, Taxi and Retweet showing the fitness of the Mamba architecture learning the parameters through MLE. with every model having similar standard deviations on log-likelihood losses with the average of about 0.02 excluding MambaTPP, THP and RMTPP having large standard deviations with the average of 2.2731 which is potentially due to the sampling method of the EasyTemporalPointProcess framework[29] having bottlenecks on sampling event sequence datasets that have little event types and long sequences as the integral sampler of these models are based on the implementation of EasyTPP which is based on the work from Neural Hawkes Processes[19] which using the original thinning algorithm where AttNHP is sampled from the original implementation which samples time with adaptive thinning. It could also possibly be caused by the intensity being calculated based on the traditional self-exciting function from Equation 2.1 with base and decay factors, where the implementation of AttNHP does not. As MambaTPP is also being implemented based on the methods from AttNHP, I conducted a direct comparison of these two models using the same implementation. The hyperparameters are still set the same with the same implementations of model components except for how intensities are being derived and the adaptive thinning algorithm.

On using the intensity modelling method and the thinning method from Yang et al.[30] showing in Table 4.6, all metrics on the StackOverflow dataset indeed get improved and MambaTPP is showing more robust results compared with the original way as all standard deviations get significantly reduced. Notably, the accuracy of MambaTPP in predicting all of the datasets has also shown improvements from 0.3% to 4.7%, which gives a large improvement on not only the StackOverflow dataset but also all other datasets.

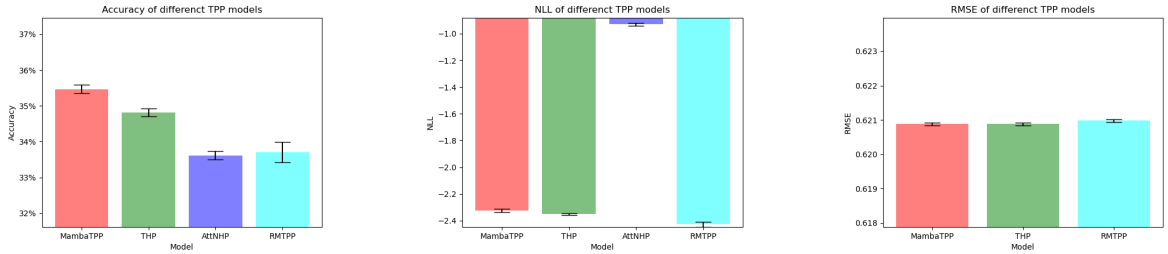

Figure 4.1: The Accuracy, log-likelihood and RMSE of the models on the Amazon dataset.

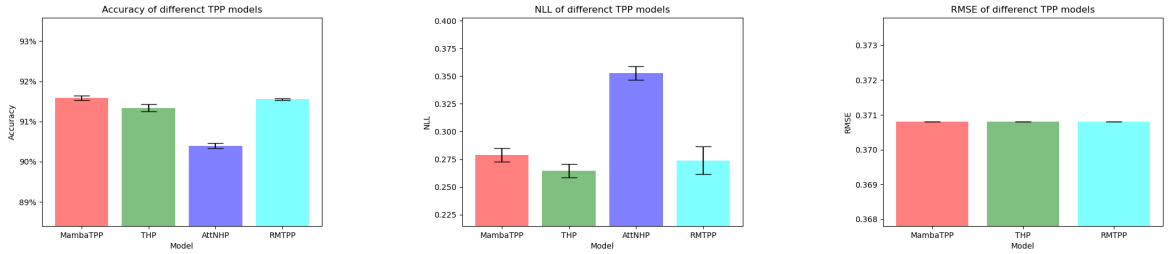

Figure 4.2: The Accuracy, log-likelihood and RMSE of the models on the Taxi dataset.

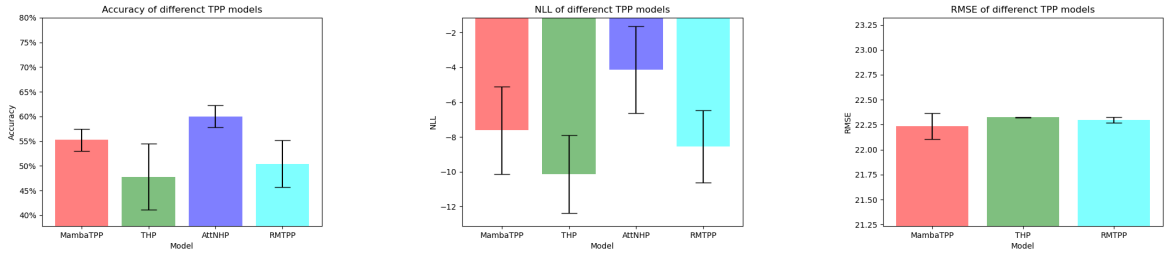

Figure 4.3: The Accuracy, log-likelihood and RMSE of the models on the Retweet dataset.

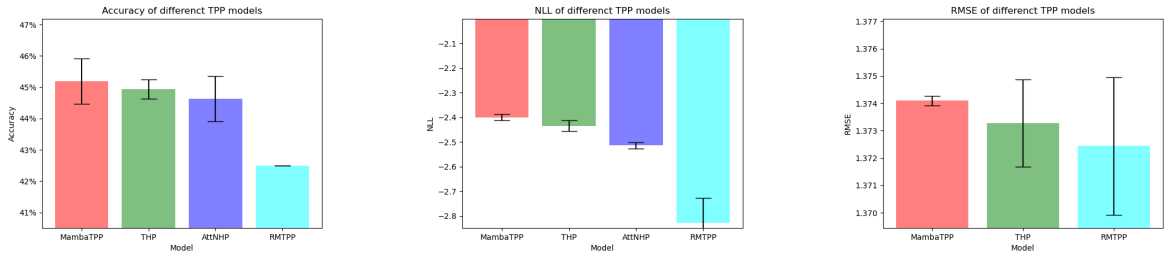

Figure 4.4: The Accuracy, log-likelihood and RMSE of the models on the StackOverflow dataset.

4.7 Comparison of efficiency on Training

As MambaTPP can potentially improve the efficiency of neural TPPs, the training time and the inference time of the attention-based models on every dataset are recorded in Table 4.7. These times are recorded with the same setting on model comparison but the batch size is set to 1 to get the time on processing per sequence. Compared with two other models MambaTPP achieved fast inference time which is close to THP and about 6 times faster than AttNHP. Also,

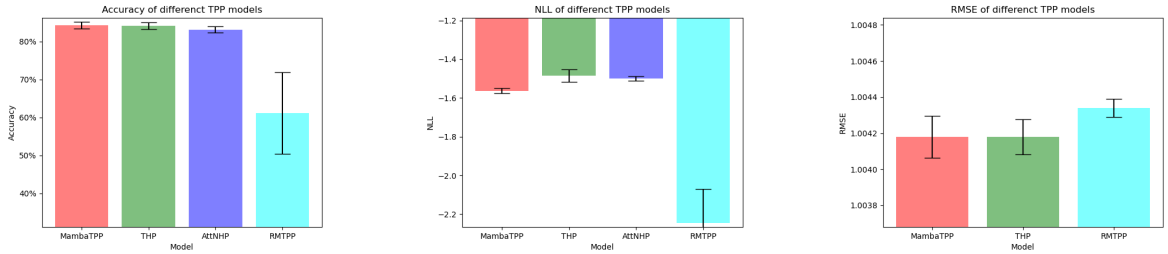

Figure 4.5: The Accuracy, log-likelihood and RMSE of the models on the MIMIC-II dataset.

Accuracy and the Standard Deviations of Accuracy of All Models					
Model	Amazon	StackOverflow	Taxi	Retweet	MIMIC-II
MambaTPP	35.46%/ 0.0012	45.18%/ 0.0072	91.59%/ 0.0006	55.27%/ 0.0225	84.30%/ 0.0082
THP	34.82%/ 0.0011	44.93%/ 0.0031	91.34%/ 0.0009	47.79%/ 0.0668	84.18%/ 0.0093
AttNHP	33.61%/ 0.0104	44.64%/ 0.0001	90.40%/ 0.0058	59.37%/ 0.0005	83.14%/ 0.0127
RMTPP	33.71%/ 0.0028	42.50%/ 0.0000	91.56%/ 0.0002	50.43%/ 0.0474	61.16%/ 0.1079

Table 4.3: The Accuracy and the standard deviation of the accuracy of Neural TPP Models on five real-world datasets.

Log-Likelihood losses and their Standard Deviation of All Models					
Model	Amazon	StackOverflow	Taxi	Retweet	MIMIC-II
MambaTPP	-2.3245/ 0.0119	-2.4002/ 0.0123	0.2790/ 0.0062	-7.6138/ 2.5080	-1.5631/ 0.0123
THP	-2.3418/ 0.0072	-2.4347/ 0.0217	0.2645/ 0.0060	-10.1280/ 2.2341	-1.4855/ 0.0326
AttNHP	-0.9295/ 0.4520	-2.5136/ 0.0068	0.3528/ 0.0226	-4.1309/ 0.0000	-1.5001/ 0.0291
RMTPP	-2.4282/ 0.0174	-2.8289/ 0.1027	0.2739/ 0.0126	-8.5309/ 2.0774	-2.2468/ 0.1768

Table 4.4: The log-likelihood losses and standard deviation of Neural TPP Models on five real-world datasets

although MambaTPP uses approximately 4 times spends training time per sequence compared with THP, it can offer better quality to match up with AttNHP using only 4-5 times less spent on to train on most of the prediction tasks which is proved by the previous benchmark. As Mamba is parallelizable with the parallel scan mechanism, which can decrease the time complexity of the Mamba layers to $O(N/T)$ where T is the number of threads, and the thinning algorithm is also parallelizable, the inference time of MambaTPP would be even faster.

RMSE and Standard Deviation of All Models					
Model	Amazon	StackOverflow	Taxi	Retweet	MIMIC-II
MambaTPP	0.6209/ 0.0002	1.3741/ 0.0002	0.3708/ 0.0000	22.2338/ 0.1294	1.0042/ 0.0001
THP	0.6209/ 0.0000	1.3733/ 0.0016	0.3708/ 0.0000	22.3232/ 0.0044	1.0042/ 0.0001
AttNHP	N.A. / N.A.	N.A. / N.A.	N.A. / N.A.	N.A. / N.A.	N.A. / N.A.
RMTPP	0.6210/ 0.0000	1.3724/ 0.0025	0.3708/ 0.0000	22.2992/ 0.0284	1.0043/ 0.0000

Table 4.5: The RMSE and their standard deviations of Neural TPP Models on five real-world datasets.

Accuracy and Log-Likelihood between MambaTPP and AttNHP					
Model	Amazon	StackOverflow	Taxi	Retweet	MIMIC-II
MambaTPP					
Accuracy	35.89%/ 0.0011	45.44%/ 0.0006	91.74%/ 0.0005	60.01%/ 0.0004	84.18%/ 0.0043
Log-Likelihood	-2.3150/ 0.0014	-2.3772/ 0.0063	0.2765/ 0.0012	-4.1356/ 0.0007	-1.5217/ 0.0356
AttNHP					
Accuracy	33.61%/ 0.0104	44.64%/ 0.0001	90.40%/ 0.0058	59.37%/ 0.0005	83.14%/ 0.0127
Log-Likelihood	-0.9295/ 0.4520	-2.5136/ 0.0068	0.3528/ 0.0226	-4.1309/ 0.0000	-1.5001/ 0.0291

Table 4.6: The accuracy and log-likelihood with their standard deviations between MambaTPP and AttNHP on five real-world datasets. Upper: Accuracy or Log-likelihood. Lower: Standard deviations of the metrics.

Training and Inference Time per Sequence(In Milliseconds)			
Dataset	MambaTPP	THP	AttNHP
Amazon	5.83/ 1.73	1.96/ 1.22	27.49/ 13.04
MIMIC	5.96/ 2.54	2.07/ 1.14	24.99/ 10.30
Taxi	5.90/ 1.63	1.97/ 0.83	23.95/ 10.47
StackOverflow	5.89/ 1.80	1.94/ 0.90	27.51/ 15.72
Retweet	4.69/ 1.62	2.24/ 1.48	25.65/ 12.40

Table 4.7: The Training time and the inference time of MambaTPP and attention-based models. Left: Training time per sequence; Right: Inference time per sequence.

Chapter 5

Conclusion and Discussion

5.1 Chapter Overview

This chapter gives a conclusion of the project with a summary of the work, the achievements and what can potentially be improved in future neural TPP works.

5.2 Summary

In this project, I have implemented a neural TPP model based on Mamba architecture, which is a selective SSM that can achieve linear time inference time. To make Mamba suitable for event sequence tasks, I implemented the embedding techniques with an embedding layer for the event type sequences. I used an augmented positional embedding technique for time sequence, which was originally used on Transformer-based models. On the model pipeline, I matched the procedures with most previous works of neural TPPs, with first preprocessing event sequence data into the format with 3 sequences on type sequence, time sequence and inter-event time sequence. Then the sequence data are embedded and concatenated into embedded history and sent into MambaTPP. By sampling log-likelihood loss using the Monte Carlo method, the NLL loss is used for MLE to learn the parameters. On event sequence inference, given the embedded history, MambaTPP samples the intensities using the thinning algorithm, where first capturing the intensity upper bound for all categorical marks and samples through the time sampling interval to get the accepted intensities. The predicted event types and event times can therefore be predicted from the event type and times of the earliest accept times. Testing my model on five event sequence datasets with three neural TPPs, MambaTPP reaches their accuracy, RMSE and log-likelihood on most of these datasets, even outperforming these models on accuracy of 0.8% to 10% approximately with direct intensity modelling.

5.3 Achievements

In this project, the key aims and goals that have been achieved are:

- As the previous literature uses traditional RNNs, LSTMs and Transformers in the implementations of neural TPPs, I have implemented the first neural TPP model using Mamba architecture, the novel Deep Learning architecture based on discretized state space models originally used on signal control for event sequence prediction. With using Mamba

architecture the inference time complexity can be decreased into linear and is also parallelizable in inference mode like other Transformer-based neural TPPs.

- Implement both methods of modelling intensities by explicitly computing intensities by adding the current, history and base influences for the intensity, and by implicitly considering the model output as the intensity. The Monte Carlo method on log-likelihood approximation and both original and adaptive thinning algorithms are implemented as well to match up state-of-the-art neural TPPs.

- Training MambaTPP using various training techniques on hyperparameter tuning with k-fold cross-validations, early stopping to prevent models from overfitting. By setting the Transformer Hawkes Process, Attentive Neural Hawkes Process and MambaTPP with nearly the same number of parameters, with Recurrent Marked Temporal Point Process, an RNN-based neural TPP, the comparison is established among them on the metrics to evaluate the model performance with these training techniques and on time consumption. Also, the time comparison is recorded to compare the efficiency between Transformer-based models and MambaTPP.

The key findings of this project are:

- MambaTPP can achieve state-of-the-art performance with both intensity modelling applications. By computing the intensity using the traditional way with current influence, history influence and base influence, MambaTPP can reach and even outperform the state-of-the-art models and show robustness to both short and long-sequence data, which shows the ability of MambaTPP to address long-term and short-term dependencies.

- With the implementation by directly computing the intensity of the function matching up with AttNHP, MambaTPP has even shown alight accurate and robust results with high efficiency in both training and inference, which is faster than AttNHP 4 to 5 times.

- The intensity modelling method by adding up current, history and base influences is potentially not good at predicting long sequence datasets with small numbers of event types as the high standard deviations on metrics of each model on the Retweet dataset. However, by changing to the direct intensity modelling method, the model shows great improvements in log-likelihood and accuracy.

5.4 Challenges and Future Works

Based on the evaluation result which is shown in Section 4.6 and Section 4.7, the challenges and future works of MambaTPP and neural TPPs are:

- Methods on temporal embedding: Currently, MambaTPP uses the positional embedding technique based on the works of Transformer-based models and augments the embedding based on the statistics of the training dataset. However, the original implementation of Mamba architecture does not utilise the positional embedding as it was designed for sequences that have evenly spaced position intervals such as text modelling where the positions are positive integers and time series forecasting with the same sizes on time intervals. Therefore it is still unknown whether other temporal embedding methods such as training a temporal embedding layer can potentially improve the performance of MambaTPP to make it learn the temporal data better.

- Event sequence datasets: Although there are various datasets provided in the area of neural TPP studies, it is still a challenge to find datasets to benchmark them. All datasets used for benchmarking were captured or preprocessed a decade ago, and datasets such as MIMIC-II have obvious patterns in this preprocessed version, which makes it challenging to measure whether the neural TPPs can be discovered in complex decision-making domains. Also, it is still difficult to have datasets that have enough quality to be preprocessed as the source of data with events and relative time due to the size of the contexts which have few or enormous numbers of event types or with extremely short sequences, concerns of user privacy, data authenticity and the difficulty of preprocessing the data into the appropriate formats.

- Combinations and augmentations with other components or mechanisms: there are still sufficient methods that can potentially improve the performance of MambaTPP. For example, in the work of AttNHP from Yang et al.[30], the implementation of Attentive Neural Datalog Through Time has combined the symbolic system which has a database to update or query, with the neural TPPs and shown great accuracy improvement on datasets that have large numbers of event types where the original neural TPPs have shown poor results on accuracy from close to 0% to 10%, and the Attentive Neural Datalog Through Time improved the accuracy to nearly 60%[30]. Also, MambaTPP has the potential to implement structural methods by letting a single MambaTPP process all event sequences and implementing a graph method to match the similarities between event sequence data not only on the event pairs in a single sequence which has also shown great enhancement on minimizing NLL and event prediction accuracy. Moreover, it is possible to combine the neural TPPs with large language models such as the work by Shi et al.[26] has shown the ability of reasoning of large language models and outperform state-of-the-art neural TPPs.

- Methods of faster sampling: Although MambaTPP has improved the efficiency during training and inference with the linear time complexities and parallelizability on recurrent mode on inference, the main time-consuming methods, which are the sampling methods of Monte Carlo method for integral estimation and thinning algorithm on event time prediction are hard to be accelerated except parallelization. It is absolutely vital in not only the area of neural TPP but also the works using sampling to estimate the results for faster sampling methods that can indeed decrease the time spent on sampling.

5.5 Conclusion

To sum up, the Mamba Temporal Point Process model is the neural temporal point process model which processes event sequence data with continuous time information. Based on the implementation of the Mamba architecture, a Deep Learning architecture based on the Selective Structured State Space Models with discretization and selection mechanism on parameters, makes it suitable for sequence modelling tasks and the ability to address long-term dependencies of the data, In this project the MambaTPP has shown ability to address long-term and short-term dependencies of event sequence data which is shown in comparison with 3 other neural temporal point processes. However, there is still an open area for better methods of time sequence embedding techniques, combinations of other event sequence modelling methods and more efficient sampling approaches.

Bibliography

[1] L. Biewald. Experiment tracking with weights and biases, 2020. Software available from wandb.com.

[2] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling, 2014.

[3] N. Du, H. Dai, R. Trivedi, U. Upadhyay, M. Gomez-Rodriguez, and L. 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*, KDD '16, page 1555–1564, New York, NY, USA, 2016. Association for Computing Machinery.

[4] J. Enguehard, D. Busbridge, A. Bozson, C. Woodcock, and N. Y. Hammerla. Neural temporal point processes for modelling electronic health records, 2020.

[5] D. Y. Fu, T. Dao, K. K. Saab, A. W. Thomas, A. Rudra, and C. Ré. Hungry hungry hippos: Towards language modeling with state space models, 2023.

[6] X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In *International Conference on Artificial Intelligence and Statistics*, 2010.

[7] A. Graves. *Supervised sequence labelling with recurrent neural networks*. Springer Berlin, 2014.

[8] A. Gu and T. Dao. Mamba: Linear-time sequence modeling with selective state spaces. *arXiv preprint arXiv:2312.00752*, 2023.

[9] A. Gu, T. Dao, S. Ermon, A. Rudra, and C. Ré. Hippo: Recurrent memory with optimal polynomial projections. *CoRR*, abs/2008.07669, 2020.

[10] A. Gu, K. Goel, and C. Ré. Efficiently modeling long sequences with structured state spaces, 2022.

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

[12] D. Hendrycks and K. Gimpel. Bridging nonlinearities and stochastic regularizers with gaussian error linear units. *CoRR*, abs/1606.08415, 2016.

[13] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization, 2017.

[14] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation Applied to Handwritten Zip Code Recognition. *Neural Computation*, 1(4):541–551, 12 1989.

[15] J. Lee, D. Scott, M. Villarroel, G. Clifford, M. Saeed, and R. Mark. Open-access mimic-ii database for intensive care research. *Proceedings of the Annual International Conference of the IEEE Engineering in Medicine and Biology Society, EMBS*, pages 8315–8318, 2011.

[16] J. Leskovec and A. Krevl. SNAP Datasets: Stanford large network dataset collection, 2014.

[17] P. A. W. Lewis and G. S. Shedler. Simulation of nonhomogeneous poisson processes by thinning. *Naval Research Logistics Quarterly*, 26(3):403–413, 1979.

[18] J. Ma, F. Li, and B. Wang. U-mamba: Enhancing long-range dependency for biomedical image segmentation, 2024.

[19] H. Mei and J. Eisner. The neural hawkes process: A neurally self-modulating multivariate point process, 2017.

[20] J. Ni. Amazon review data, 2018.

[21] R. Pascanu, T. Mikolov, and Y. Bengio. On the difficulty of training recurrent neural networks, 2013.

[22] P. Ramachandran, B. Zoph, and Q. V. Le. Searching for activation functions, 2017.

[23] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *Nature*, 323:533–536, 1986.

[24] F. P. Schoenberg. Introduction to point processes. 2011.

[25] O. Shchur, A. C. Türkmen, T. Januschowski, and S. Günnemann. Neural temporal point processes: A review, 2021.

[26] X. Shi, S. Xue, K. Wang, F. Zhou, J. Y. Zhang, J. Zhou, C. Tan, and H. Mei. Language models can improve event prediction by few-shot abductive reasoning. In *Advances in Neural Information Processing Systems*, 2023.

[27] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 6000–6010, Red Hook, NY, USA, 2017. Curran Associates Inc.

[28] C. Whong. FOILing NYC's taxi trip data, 2014.

[29] S. Xue, X. Shi, Z. Chu, Y. Wang, H. Hao, F. Zhou, C. Jiang, C. Pan, J. Y. Zhang, Q. Wen, J. Zhou, and H. Mei. Easytpp: Towards open benchmarking temporal point processes. In *International Conference on Learning Representations (ICLR)*, 2024.

[30] C. Yang, H. Mei, and J. Eisner. Transformer embeddings of irregularly spaced events and their participants, 2022.

[31] Q. Zhang, A. Lipani, O. Kirnap, and E. Yilmaz. Self-attentive Hawkes process. In H. D. III and A. Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 11183–11193. PMLR, 13–18 Jul 2020.

[32] Q. Zhang, A. Lipani, O. Kirnap, and E. Yilmaz. Self-attentive hawkes processes, 2020.

[33] K. Zhou, H. Zha, and L. Song. Learning triggering kernels for multi-dimensional hawkes processes. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2013.

[34] L. Zhu, B. Liao, Q. Zhang, X. Wang, W. Liu, and X. Wang. Vision mamba: Efficient visual representation learning with bidirectional state space model, 2024.

[35] S. Zuo, H. Jiang, Z. Li, T. Zhao, and H. Zha. Transformer Hawkes process. In H. D. III and A. Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 11692–11702. PMLR, 13–18 Jul 2020.

[36] S. Zuo, H. Jiang, Z. Li, T. Zhao, and H. Zha. Transformer hawkes process, 2021.

Appendix A

1

A.1 Hyperparameter tuning details

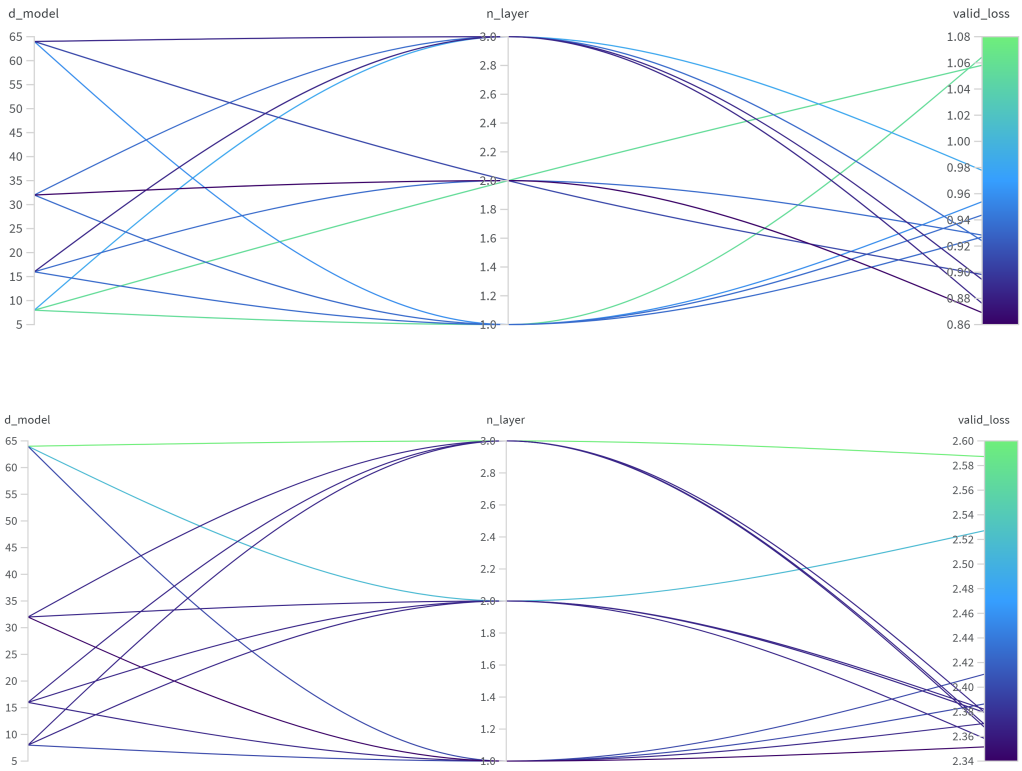

Figure A.1: The validation NLL loss chart of hyperparameter tuning of MambaTPP on MIMIC-II and StackOverflow. Upper: Hyperparameter tuning on MIMIC-II dataset. Lower: Hyperparameter tuning on StackOverflow dataset.