Learning Point Processes using Recurrent Graph Network: Supplementary Material

Theorem 1 (Time-Rescaling Theorem): Let $0 < t_1 < t_2, ..., t_N < T$ be a realization from a point process with conditional intensity function $\lambda(t|\mathcal{H}_t)$, define

$$z_j = \int_{t_{j-1}}^{t_j} \lambda(t|\mathcal{H}_t)\, dt \tag{1}$$

for $j = 1, ..., N$ and $t_0 = 0$. Then, z_j are independent exponentially distributed random variables with rate parameter 1.

I. Appendix A

II. Monte Carlo Estimate of the Integral

$$\hat{\Lambda}_{\text{MC}} = \sum_{j=1}^{L_i} (t_j - t_{j-1}) \left(\frac{1}{N_\tau} \sum_{k=1}^{N_\tau} \lambda(\tau_k) \right) \tag{2}$$

where, $\tau_k \sim \text{U}(t_{j-1}, t_j)$ and N_τ is the number of points used for evaluation. This allows for an unbiased estimate of the integral as $\mathbb{E}(\hat{\Lambda}_{\text{MC}}) = \Lambda$ (Proof in supplementary materials). Similarly, the gradients can be approximated using:

$$\nabla\hat{\Lambda}_{\text{MC}} = \sum_{j=1}^{L_i} (t_j - t_{j-1}) \left(\frac{1}{N_\tau} \sum_{k=1}^{N_\tau} \nabla\lambda(\tau_k) \right) \tag{3}$$

$$\hat{\Lambda}_{\text{MC}} = \sum_{j=1}^{L_i} (t_j - t_{j-1}) \left(\frac{1}{N_\tau} \sum_{k=1}^{N_\tau} \lambda(\tau_k) \right) \tag{4}$$

$$\mathbb{E}[\hat{\Lambda}_{\text{MC}}] = \sum_{j=1}^{L_i} (t_j - t_{j-1}) \left(\frac{1}{N_\tau} \sum_{k=1}^{N_\tau} \mathbb{E}[\lambda(\tau_k)] \right) \tag{5}$$

As $\tau_k \in (t_{j-1}, t_j)$ are i.i.d, they have the same expectation $\mathbb{E}[\tau_j]$.

$$\mathbb{E}[\hat{\Lambda}_{\text{MC}}] = \sum_{j=1}^{L_i} (t_j - t_{j-1}) \left(\mathbb{E}[\lambda(\tau_j)] \right) \tag{6}$$

The support of $\mathbb{E}[\tau_j]$ lies in (t_{j-1}, t_j),

$$\mathbb{E}[\lambda_j] = \int_{t_{j-1}}^{t_j} \frac{1}{(t_j - t_{j-1})} \lambda(s)\, ds \tag{7}$$

substituting in Eq. 6,

$$\mathbb{E}[\hat{\Lambda}_{\text{MC}}] = \sum_{j=1}^{L_i} (t_j - t_{j-1}) \left(\int_{t_{j-1}}^{t_j} \frac{1}{(t_j - t_{j-1})} \lambda(s)\, ds \right) \tag{8}$$

Table I: Dataset statistics

Dataset	$\lvert\mathcal{Y}\rvert$	Seq. Length	# of Seq.		
		Mean	Train	Val	Test
Retweets	3	109	20000	2000	2000
StackOverflow	22	72	4777	530	1326
MIMIC-II	75	4	527	58	65
Financial	2	2074	90	10	100
StarCraft II	16	78	6141	1316	1317

simplifying,

$$\mathbb{E}[\hat{\Lambda}_{\text{MC}}] = \sum_{j=1}^{L_i} \int_{t_{j-1}}^{t_j} \lambda(s)\, ds \tag{9}$$

$$\mathbb{E}[\hat{\Lambda}_{\text{MC}}] = \int_0^T \lambda(s)\, ds = \Lambda \qquad \blacksquare \tag{10}$$

Thus, the Monte Carlo approximation is unbiased.

III. Implementation Details

Dropout [1] probability of 0.1 is used to reduce overfitting. We also make use of Layer Normalization in the input-to-graph update and edge update stages to prevent explosion of gradients and activations to stabilize training. ADAM is used as the optimizer [2] and the learning rate is chosen to be 10^{-4}. The final complete loss function to be minimized is:

$$\mathcal{L} = \mathcal{L}_\lambda + \beta_y \mathcal{L}_y + \beta_t \mathcal{L}_t \tag{11}$$

In the loss function $\mathcal{L}$, β_y is set to 1 while β_t is set to 100 to scale the L2 error. The hyper-parameters for various datasets are shown in table II. The learning rate was 10^{-4} for all the datasets and the optimizer used was ADAM [2]. LeakyReLU parameter was set to $\alpha = 0.2$. The model was trained till 50 epochs for all the datasets. We stack 2 layers of GAT for graph-to-graph update. Additionally, we add a residual connection that allows for skipping through the GAT Layer for better flow of gradients

Table II: Hyper-parameters

Dataset	$d_{in} = d_v = d_u$	d_e	# Heads	TBPTT steps
Retweets	256	16	8	20
StackOverflow	256	16	16	20
Financial	256	16	16	20
MIMIC II	128	16	8	20
StarCraft II	256	32	8	40

The hyper-parameters for THP were the ones prescribed by the authors and the official github implementation was used for results. For StarCraft II, the model parameters

were $M = 256$, $K = V = 64$, $d_{inner} = 2048$, # Heads = 8.

For RMTPP, NHP and THP, instead of performing $\int_{t_j}^{\infty} t\, p(t|\mathcal{H}_t)\, dt$ to find the time of the next event, we add a FC layer to the models to directly predict the time of the next event $\hat{t}$ using the latent embeddings. We modify RMTPP to use the same λ^* expression as ours instead of the original one proposed by the authors as it restricted the conditional intensity expression to the exponential family for a tractable likelihood computation. This also allows for a fair comparison between the models. This is not done for NHP because NHP allows for latent embeddings to be generated at any time t, thus we have used the one proposed by the authors.

IV. StarCraft II Dataset

StarCraft II is a real time strategy game in which players play against each with the objective of building an army and defeating the opponent. The vast event space of the game leads to significant complexity and has led to the game becoming a popular AI testbed. The large amount of available human replay files make it possible to create a suitable dataset for this work. To achieve dataset creation, we build a data generation tool based on the open source PySC2 environment and the StarCraft II machine learning API supported by the game developer. The generation tool runs the original game environment and an observer agent that gathers data from the game environment through the PySC2 interface. It performs data extraction based on specific observation rules defined by the user to generate data accordingly. As shown in Fig. 1, for the dataset used in this work, we define an observation rule that records certain player actions as event sequences from observing the game environment running the replay files of one versus one games.

There are different types of buildings in StarCraft II and each serves different purpose. For example, there are buildings that produce army and buildings that develop technology. The location of buildings also has important influences. For example, players can choose to put buildings in clusters in order to more effectively protect them or they can choose to put buildings in separate locations so that the opponent has more difficulties in finding them. Therefore, the strategy that a player adopts can be largely reflected in the building construction information. In addition, the evolution of strategy taken by each player has a dynamic that depends on interaction with its opponent. The correlation between two players' s actions thus contains important information about their strategies.

Based on the preceding discussion, for data used in this work, we choose to extract sequences of building constructions. To achieve this, the observation rule is defined with a list of action-of-interests related to building constructions. The observer agent extracts building construction actions of both players in the game. It records three dimensions of each actions: building type, building time and building location. Each extracted data point contains two sequences: one for each player.

A part of a game replay used in the event-to-event sequence is illustrated in Fig. 2: the player construct a Pylon type building at the beginning, which provides resources for building more units. This is followed by a Gateway building for army production at time of 1:42. Later in the game, the player build a Nexus, which is used for resource gathering, at a remote location. This move is performed to expand the occupied territory. It is also worth noting that the shown visualization is for demonstration purpose only, and the data extraction tool as described does not rely on the fully rendered game. It extracts information from the API directly to improve data generation efficiency.

References

[1] N. Srivastava *et al.*, "Dropout: A simple way to prevent neural networks from overfitting," *JMLR*, 2014.

[2] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *ICLR 2015*.

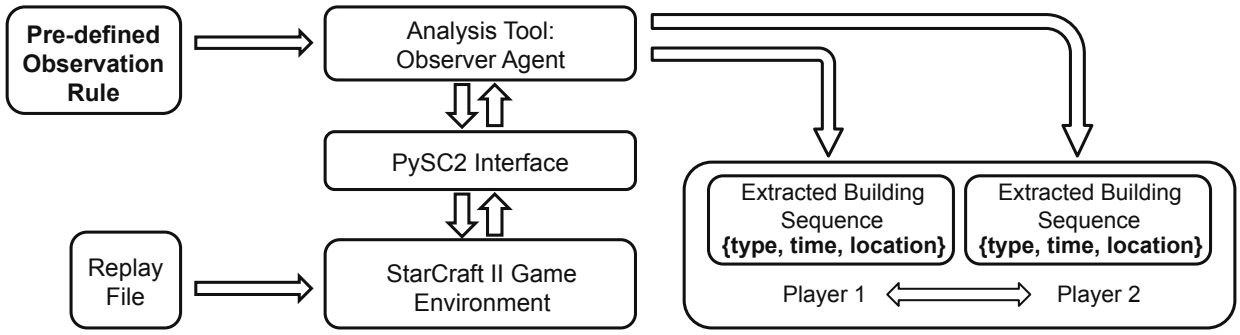

Figure 1: Data Generation Tool

Figure 2: Example Sequence