PoPPy: A Point Process Toolbox Based on PyTorch

Hongteng Xu
Infinia ML, Inc. and Duke University
hongtengxu313@gmail.com

1 Overview

1.1 What is PoPPy?

PoPPy is a Point Process toolbox based on PyTorch, which achieves flexible designing and efficient learning of point process models. It can be used for interpretable sequential data modeling and analysis, e.g., Granger causality analysis of multi-variate point processes, point process-based simulation and prediction of event sequences.

1.2 The Goal of PoPPy

Many real-world sequential data are often generated by complicated interactive mechanisms among multiple entities. Treating the entities as events with different discrete categories, we can represent their sequential behaviors as event sequences in continuous time domain. Mathematically, an event sequence \( s \) can be denoted as \( \{ (t^s_i, c^s_i, f^c c^s_i) \}_{i=1}^L \), where \( t^s_i \) and \( c^s_i \) are the timestamp and the event type (i.e., the index of entity) of the \( i \)-th event, respectively. Optionally, each event type may be associated with a feature vector \( f^c \in \mathbb{R}^{D^c}, c \in C \), and each event sequence may also have a feature vector \( f^s \in \mathbb{R}^{D^s}, s \in S \). Many real-world scenarios can be formulated as event sequences, as shown in Table 1.

| Task                | Scene                      | Patient admission              | Job hopping                  | Online shopping               |
|---------------------|----------------------------|--------------------------------|------------------------------|-------------------------------|
| Entities (Event types) | Diseases                  | Patients’ admission records | LinkedIn users’ job history | Items                        |
| Sequences           | Diagnose records          | Patient profiles              | Job descriptions             | Buying/rating behaviors       |
| Event feature       | Build Disease network     |                                | User profiles                | Item profiles                 |
| Sequence feature    |                            |                                | Model talent flow            | User profiles                 |
| Task                |                            |                                |                              | Recommendation system         |

Table 1: Typical event sequences in practice.

Given a set of event sequences \( S \), we aim to model the dynamics of the event sequences, capture the interactive mechanisms among different entities and predict their future behaviors. Temporal point process model provides us with a potential solution to achieve these aims. In particular, a multi-variate temporal point process can be represented by a set of counting processes \( N = \{ N_c(t) \}_{c \in C} \), in which \( N_c(t) \) is the number of type-\( c \) events occurring till time \( t \). For each \( N_c(t) \), the expected instantaneous

Preprint. Work in progress.
The happening rate of type- \( c \) events at time \( t \) is denoted as \( \lambda_c(t) \), which is called “intensity function”:

\[
\lambda_c(t) = \frac{\mathbb{E}[dN_c(t)|\mathcal{H}_t]}{dt}, \quad \mathcal{H}_t = \{(t_i, c_i)|t_i < t, c_i \in C\},
\]

where \( \mathcal{H}_t \) represents historical observations before time \( t \).

As shown in Fig. 1, the counting processes can be represented as a set of intensity functions, each of which corresponds to a specific event type. The temporal dependency within the same event type and that across different event types (i.e., the red arrows in Fig. 1) can be captured by choosing particular intensity functions. Therefore, the key points of point process-based sequential data modeling include:

1. How to design intensity functions to describe the mechanism behind observed data?
2. How to learn the proposed intensity functions from observed data?

The goal of PoPPy is providing a user-friendly solution to the key points above and achieving large-scale point process-based sequential data analysis, simulation and prediction.

### 1.3 Installation of PoPPy

PoPPy is developed on Mac OS 10.13.6 but also tested on Ubuntu 16.04. The installation of PoPPy is very simple. In particular,

1. Install Anaconda3 and create a conda environment.
2. Install PyTorch0.4 in the environment.
3. Download PoPPy from [https://github.com/HongtengXu/PoPPy/](https://github.com/HongtengXu/PoPPy/) and unzip it to the directory in the environment. The unzipped folder should contain several subfolders, as shown in Fig. 2.
4. Open `dev/util.py` and change `POPPY_PATH` to the directory, as shown in Fig. 3.

Figure 1: Event sequences and intensity functions.

Figure 2: The subfolders in the package of PoPPy.

The subfolders in the package include:

- **data**: It contains a toy dataset in `.csv` format.
- **dev**: It contains a `util.py` file, which configures the path and the logger of the package.
- **docs**: It contains the tutorial of PoPPy.

2
PoPPy provides three functions to load data from .csv file and convert it to the proposed database.
2.1.1 \texttt{preprocess.DataIO.load\_sequences\_csv}

This function loads event sequences and converts them to the proposed \texttt{database}. The IO and the description of this function is shown in Fig. 4.

```python
@staticmethod
def load_sequences_csv(file_name: str, domain_names: Dict) -> Dict:
    """Load event sequences from a csv file
    :param file_name: the path and name of the target csv file
    :param domain_names: a dictionary contains the names of the key columns corresponding to ('seq_id', 'time', 'event')
    The format should be:
    domain_names = {'seq_id': the column name of sequence name, 'time': the column name of timestamps, 'event': the column name of events}
    :return: database: a dictionary containing observed event sequences
    database = {'event_features': None,
                'type2id': a Dict = {'event_name': event_index}
                'id2type': a Dict = {event_index: 'event_name'}
                'idx2id': a Dict = {'seq_name': seq_index}
                'idx2type': a Dict = {'seq_name': seq_name}
                'sequences': a List = [seq_i, seq_2, ..., seq_N].
    }
    
    For the i-th sequence:
    seq_i = {'times': (N), float array of timestamps, N is the number of events.
    'events': (N), int array of event types.
    'seq_feature': None.
    't_start': a float number, the start timestamp of the sequence.
    't_stop': a float number, the stop timestamp of the sequence.
    'label': None}
```

Figure 4: The description of \texttt{load\_sequences\_csv}.

For example, the \texttt{Linkedin.csv} file in the folder \texttt{data} records a set of linkedin users’s job hopping behaviors among different companies, whose format is shown in Fig. 5.

|   | id | time | event          | option          |
|---|----|------|----------------|-----------------|
| 1 | 21 | 38.79| Google Intern  | Research        |
| 1 | 39 | 29521| UC Berkeley    | Graduate Student Research |
| 1 | 33 | 34.0849| Google        | Sr Research Science |
| 6 | 29 | 30   | Google         | Software Eng Intern |
| 6 | 30 | 43   | Google         | Software Eng Intern |
| 6 | 32.2027| Google  | Software Eng Intern |
| 6 | 32.3541| Google  | Software Eng Intern |
| 6 | 32.5181| Baidu   | Software Eng   |
| 6 | 33.2048| Google  | Software Eng   |
| 6 | 33.5068| Microsoft| Software Eng Intern |
| 8 | 31 | Linkin | Software Eng Intern |
| 8 | 32.0849| Linkin  | Software Eng Intern |
| 8 | 32.9178| UCLA    | Graduate Teaching Assistant |

Figure 5: Some rows of \texttt{Linkedin.csv}.

Here, the column \texttt{id} corresponds to the names of sequences (i.e. the index of users), the column \texttt{time} corresponds to the timestamps of events (i.e. the ages that the users start to work), and the column \texttt{event} corresponds to the event types (i.e., the companies). Therefore, we can define the \texttt{input domain_names} as

```python
domain_names = {
    'seq_id': 'id',
    'time': 'time',
    'event': 'event'
}
```

```python
and database = load_sequences_csv('Linkedin.csv', domain_names).
```
Note that the database created by `load_sequences_csv()` does not contain event features and sequence features, whose values in `database` are `None`. PoPPy supports to load categorical or numerical features from `.csv` files, as shown below.

### 2.1.2 `preprocess.DataIO.load_seq_features_csv`

This function loads sequence features from a `.csv` file and import them to the proposed `database`. The IO and the description of this function is shown in Fig. 6. Take the `Linkedin.csv` file as an example.

```python
def load_seq_features_csv(file_name: str, seq_domain: str, domain_dict: Dict, database: Dict, normalize: int=0):
    load sequences' features from a csv file
    :param file_name: the path and the name of the csv file
    :param seq_domain: the name of the key column corresponding to sequence index.
    :param domain_dict: a dictionary containing the names of the key columns corresponding to the features.
    :param normalize: 0 = no normalization, 1 = normalization across features, 2 = normalization across sequences
    :return: a database having sequences' features
```

Figure 6: The description of `load_seq_features_csv`.

Suppose that we have already create `database` by the function `load_sequences_csv`, and we want to take the column `option1` (i.e., the job titles that each user had) as the categorical features of event sequences. We should have

```python
domain_names = {
    'option1': 'categorical'
}
database = load_seq_features_csv(
    file_name = 'Linkedin.csv',
    seq_domain = 'seq_id',
    domain_dict = domain_names,
    database = database
)
```

Here the input `normalize` is set as default 0, which means that the features in `database[‘sequences’][i][‘seq_feature’], i = 1,...,|S|, are not normalized.

### 2.1.3 `preprocess.DataIO.load_event_features_csv`

This function loads event features from a `.csv` file and import them to the proposed `database`. The IO and the description of this function is shown in Fig. 7. Similarly, if we want to take the column `option1` in `Linkedin.csv` as the categorical features of event types, we should have

```python
domain_names = {
    'option1': 'categorical'
}
database = load_event_features_csv(
    file_name = 'Linkedin.csv',
    event_domain = 'event',
    domain_dict = domain_names,
    database = database
)
```
2.2 Operations for Data Preprocessing

Besides basic sequence/feature loaders and converters mentioned above, PoPPy contains multiple useful functions and classes for data preprocessing, including sequence stitching, superposing, aggregating and batch sampling. Fig. 8 illustrates the corresponding data operations.

![Data Operations Illustration]

Figure 8: The illustration of four data operations.

2.2.1 `preprocess.DataOperation.stitching`

This function stitches the sequences in two `database` randomly or based on their `seq_feature` and time information (`t_start`, `t_stop`). Its description is shown in Fig. 9.

When `method = 'random'`, for each sequence in `database1` the function randomly selects a sequence in `database2` as its follower and stitches them together. When `method = 'feature'`, the similarity between the sequence in `database1` and that in `database2` is defined by the multiplication of a temporal Gaussian kernel and a sequence feature’s Gaussian kernel.
and the function selects the sequence in $\text{database2}$ yielding to a distribution defined by the similarity. The stitching method has been proven to be useful for enhancing the robustness of learning results, especially when the training sequences are very short [9, 4].

```python
def stitching(database1: Dict, database2: Dict, method: str = 'random') -> Dict:
    Stitch each sequence in $\text{database2}$ to the end of one sequence of $\text{database1}$

    :param database1: the observed event sequences
    :param database2: another observed event sequences
    :param database: an event's static features; None or (O, 1) float array of event's static features, O is the number of event types.
    :type2idx: a Dict = {event_name: event_index}
    :idx2type: a Dict = {event_index: event_name}
    :seq2idx: a Dict = {seq_name: seq_index}
    :seq2type: a Dict = {seq_index: seq_name}
    :sequences: a List = [seq_1, seq_2, ..., seq_N].

    For the i-th sequence:
    seq_i = ('times': (N,) float array of timestamps, N is the number of events.
    'events': (N,) int array of event types.
    'seq_feature': None or (O, 1) float array of sequence's static feature.
    't_start': a float number indicating the start timestamp of the sequence.
    't_stop': a float number indicating the stop timestamp of the sequence.
    'label': None or int/float number indicating the labels of the sequence)

    :param method: a string indicates stitching method:
    "random": stitch the seq_j in sequences2 to the seq_i in sequences1 for j ~ {1, ..., N, i=1, ..., N and
    when $\text{method = 'feature'}$, the similarity between the sequence in $\text{database1}$ and that in $\text{database2}$ is defined by the multiplication of a temporal Gaussian kernel and a sequence feature's Gaussian kernel, and the function selects the sequence in $\text{database2}$ yielding to a distribution defined by the similarity.

    Similar to the stitching operation, the superposing method has been proven to be useful for learning linear Hawkes process robustly. However, it should be noted that different from stitching operation, which stitches similar sequences with a high probability, the superposing operation would like to superpose the dissimilar sequences with a high probability. The rationality of such an operation can be found in my paper [8, 5].

2.2.3 preprocess.DataOperation.aggregating

This function discretizes each event sequence into several bins and counts the number of events with specific types in each bin. Its description is shown in Fig. [11]

2.2.4 preprocess.DataOperation.EventSampler

This class is a subclass of torch.utils.data.Dataset, which samples batches from $\text{database}$. For each sample in the batch, an event (i.e., its event type and timestamp) and its history with length $\text{memorysize}$ (i.e., the last $\text{memorysize}$ events and their timestamps) are recorded. If event and/or sequence features are available, the sample will record these features as well.

Figure 9: The description of stitching.
3 Temporal Point Process Models

3.1 Modular design of point process model

PoPPy applies a flexible strategy to build point process’s intensity functions from interpretable modules. Such a modular design strategy is very suitable for Hawkes process and its variants. Fig. 13 illustrates the proposed modular design strategy. In the following sections, we take Hawkes process and its variants as examples, and introduce the modules (i.e., the classes) in PoPPy.

3.2 model.PointProcess.PointProcessModel

This class contains basic functions of a point process model, including

- **fit**: learn model’s parameters given training data. It description is shown in Fig. 14.
- **validation**: test model given validation data. It description is shown in Fig. 15.
- **simulation**: simulate new event sequences from scratch or following observed sequences by Ogata’s thinning algorithm [3]. It description is shown in Fig. 16.
- **prediction**: predict expected counts of the events in the target time interval given learned model and observed sequences. It description is shown in Fig. 17.
- **model_save**: save model or save its parameters. It description is shown in Fig. 18.
- **model_load**: load model or load its parameters. It description is shown in Fig. 19.
- **print_info**: print basic information of model
- **plot_exogenous**: print exogenous intensity.

In PoPPy, the instance of this class actually implements an inhomogeneous Poisson process, in which the exogenous intensity is used as the intensity function.

An important subclass of this class is model.HawkesProcess.HawkesProcessModel. This subclass inherits most of the functions above except **print_info** and **plot_exogenous**. Additionally, because Hawkes process considers the triggering patterns among different event types, this
subclass has a new function `plot_causality`, which plots the adjacency matrix of the event types’ Granger causality graph. The typical visualization results of the exogenous intensity of different event types and the Granger causality among them are shown in Fig. 20.

Compared with its parent class, `model.HawkesProcess.HawkesProcessModel` uses a specific intensity function, which is defined in the class `model.HawkesProcess.HawkesProcessIntensity`.

### 3.3 model.HawkesProcess.HawkesProcessIntensity

This class inherits the functions in `torch.nn.Module`. It defines the intensity function of a generalized Hawkes process, which contains the following functions:

- `print_info`: print the basic information of the intensity function.
Figure 13: An illustration of proposed modular design strategy. Each color block represents a class with some functions. For each block, the dotted frame represents one of its subclass, which inherits some functions (the white ones) while overrides some others or creates new ones (the yellow ones). The black arrow means that the destination class will call the instance of the source class as input.

```
set fit(self, dataloader, optimizer, epochs: int, scheduler=None, sparsity: float=None, nonnegative=None, use_cuda: bool=False, validation_set=None):
    
    Learn parameters of a generalized Hawkes process given observed sequences
    :param dataloader: a pytorch batch-based data loader
    :param optimizer: the sgd optimization method defined by PyTorch
    :param epochs: the number of training epochs
    :param scheduler: the method adjusting the learning rate of SGD defined by PyTorch
    :param sparsity: None or a float or an int (lower bound, typically the lower bound = 0)
    :param use_cuda: use cuda (true) or not (false)
    :param validation_set: None or a validation dataloader
```

Figure 14: The description of `fit`.

- **intensity**: calculate $\lambda_{c_i}(t_i)$ of the $i$-th sample in the batch sampled by EventSampler.
- **expected_counts**: calculate $\int_{t_{i-1}}^{t_i} \lambda_c(s) ds$ for $c \in C$ and for the $i$-th sample in the batch.
- **forward**: override the forward function in torch.nn.Module. It calculates $\lambda_{c_i}(t_i)$ and $\int_{t_{i-1}}^{t_i} \lambda_c(s) ds$ for $c \in C$ for SGD.
Specifically, the intensity function of type-$c$ event at time $t$ is defined as

$$
\lambda_c(t) = g_{\lambda} \left( \mu_c(f_c, f_s) + \sum_{t_i < t} \phi_{cc_i}(t - t_i, f_c, f_{c_i}) \right)
$$

(2)

Here, the intensity function is consist of two parts:

- **Exogenous intensity** $\mu_c(f_c, f_s)$: it is independent with time, which measures the intensity contributed by the intrinsic properties of sequence and event type.

- **Endogenous impact** $\sum_{t_i < t} \phi_{cc_i}(t - t_i, f_c, f_{c_i})$: it sums up the influences of historical events quantitatively via impact functions $\{\phi_{cc'}(t)\}_{c,c' \in C}$, which measures the intensity contributed by the historical observations.
Furthermore, the impact function is decomposed with the help of basis representation, where $\kappa_m(t)$ is called the $m$-th decay kernel and $\alpha_{cc,m}(f_c, f_n)$ is the corresponding coefficient.

$g_\lambda(\cdot)$ is an activation function, which can be

- **Identity**: $g(x) = x$.
- **ReLU**: $g(x) = \max\{x, 0\}$.
- **Softplus**: $g(x) = \frac{1}{\beta} \log(1 + \exp(-\beta x))$.

PoPPy provides multiple choices to implement various intensity functions — each module can be parametrized in different ways.

### 3.3.1 model.ExogenousIntensity.BasicExogenousIntensity

This class and its subclasses in `model.ExogenousIntensityFamily` implement several models of exogenous intensity, as shown in Table 2.

Here, the activation function $g(\cdot)$ is defined as aforementioned $g_\lambda$. 
def load_model(self, full_path, mode='entire'):
    Load pre-trained model
    :param full_path: the path of directory
    :param mode: 'parameter' for saving only parameters of the model,
                 'entire' for saving entire model

Figure 19: The description of model_load.

![Figure 20: Typical visualization results.](image)

(a) exogenous intensity    (b) Granger causality

Note that the last two models require event and sequence features as input. When they are called while the features are not given, PoPPy will add one more embedding layer to generate event/sequence features from their index, and learn this layer during training.

3.3.2 model.EndogenousImpact.BasicEndogenousImpact

This class and its subclasses in model.EndogenousImpactFamily implements several models of the coefficients of the impact function, as shown in Table 3.

Here, the activation function $g(\cdot)$ is defined as aforementioned $g_{\lambda}$.

Note that the last two models require event and sequence features as input. When they are called while the features are not given, PoPPy will add one more embedding layer to generate event/sequence features from their index, and learn this layer during training.

3.3.3 model.DecayKernel.BasicDecayKernel

This class and its subclasses in model.DecayKernelFamily implements several models of the decay kernel, as shown in Table 4.

Fig. 21 visualizes some examples.

4 Learning Algorithm

4.1 Loss functions

With the help of PyTorch, PoPPy learns the point process models above efficiently by stochastic gradient descent on CPU or GPU.\(^1\) Different from existing point process toolboxes, which mainly

\(^1\) Currently, the GPU version is under development.
Table 2: Typical models of exogenous intensity.

| Class | Formulation |
|-------|-------------|
| ExogenousIntensity.BasicExogenousIntensity | $\mu_c(f_c, f_s) = \mu_c$ |
| ExogenousIntensityFamily.NaiveExogenousIntensity | $\mu_c(f_c, f_s) = g(\mu_c)$ |
| ExogenousIntensityFamily.LinearExogenousIntensity | $\mu_c(f_c, f_s) = g(w_i f_s)$ |
| ExogenousIntensityFamily.NeuralExogenousIntensity | $\mu_c(f_c, f_s) = NN(f_c, f_s)$ |

Table 3: Typical models of endogenous impact’s coefficient.

| Class | Formulation |
|-------|-------------|
| EndogenousImpact.BasicEndogenousImpact | $\alpha_{cc'}(f_c, f_{c'}) = \alpha_{cc'}$ |
| EndogenousImpactFamily.NaiveEndogenousImpact | $\alpha_{cc'}(f_c, f_{c'}) = g(\alpha_{cc'})$ |
| EndogenousImpactFamily.FactorizedEndogenousImpact | $\alpha_{cc'}(f_c, f_{c'}) = g(u^{c}_m v^{c'}_m)$ |
| EndogenousImpactFamily.LinearEndogenousImpact | $\alpha_{cc'}(f_c, f_{c'}) = g(w^{c}_m f_c)$ |
| EndogenousImpactFamily.BilinearEndogenousImpact | $\alpha_{cc'}(f_c, f_{c'}) = g(f_c^{T} W_m f_{c'})$ |

focuses on the maximum likelihood estimation of point process models, PoPPy integrates three loss functions to learn the models, as shown in Table 5.

Here $\lambda(t) = [\lambda_1(t), ..., \lambda_{|C|}(t)]$ and $1_c$ is an one-hot vector whose the $c$-th element is 1.

4.2 Stochastic gradient decent

All the optimizers and the learning rate scheduler integrated in PyTorch are applicable to PoPPy. A typical configuration is using Adam + Exponential learning rate decay strategy, which should achieve good learning results in most situations. The details can be found in the demo scripts in the folder example.

**Trick:** Although most of optimizers are applicable, generally Adam achieves the best performance in our experiments [2].

4.3 Optional regularization

Besides the L2-norm regularizer integrated in the optimizers of PyTorch, PoPPy provides two more regularizers when learning models.

1. **Sparsity:** L1-norm of model’s parameters can be applied to the models, which helps to learn structural parameters.

2. **Nonnegativeness:** If it is required, PoPPy can ensure the parameters to be nonnegative during training.

**Trick:** When the activation function of impact coefficient is softplus, you’d better close the nonnegative constraint by setting the input `nonnegative` of the function `fit` as `None`.

5 Examples

As a result, using PoPPy, users can build their own point process models by combining different modules with high flexibility. As shown in Fig. [22] Each point process model can be built by selecting different modules and combining them together. The red dots represent the module with learnable parameters, the blue dots represent the module without parameters, and the green dots represent loss function modules. Moreover, users can add their own modules and design specific point process models for their applications easily, as long as the new classes override the corresponding functions.

Finally, we list some typical models implemented by PoPPy in Table 6.

---

2 It should be noted that our implementations may be different from the methods in the references in the aspect of model and learning algorithm so the results in the references may not be reproduced by PoPPy.
Table 4: Typical models of decay kernel.

| Class                                 | $M$ | Formulation                                                                 |
|---------------------------------------|-----|------------------------------------------------------------------------------|
| DecayKernelFamily.ExponentialKernel   | 1   | $\kappa(t) = \begin{cases} \omega \exp\left(-\omega(t-\delta)\right), & t \geq \delta, \\ 0, & t < \delta \end{cases}$ |
| DecayKernelFamily.RayleighKernel      | 1   | $\kappa(t) = \omega t \exp\left(-\omega t^2/s\right)$                      |
| DecayKernelFamily.GaussianKernel      | 1   | $\kappa(t) = \frac{2}{\sqrt{2\pi} \sigma} \exp\left(-\frac{t^2}{2\sigma^2}\right)$ |
| DecayKernelFamily.PowerlawKernel      | 1   | $\kappa(t) = \begin{cases} (\omega - 1)^{-1} t^{-\omega}, & x \geq \delta, \\ (\omega - 1)/\delta, & 0 < x < \delta \end{cases}$ |
| DecayKernelFamily.GateKernel          | 1   | $\kappa(t) = \frac{1}{\delta}, \quad t \in [\omega, \omega + \delta]$     |
| DecayKernelFamily.MultiGaussKernel    | >1  | $\kappa_m(t) = \frac{1}{\sqrt{2\pi} \sigma_m} \exp\left(-\frac{(t-t_m)^2}{2\sigma_m^2}\right)$ |

(a) Exponential kernel  
(b) Rayleigh kernel   
(c) Gaussian kernel      
(d) Powerlaw kernel    
(e) Gate kernel          
(f) Multi-Gaussian kernel

Figure 21: Examples of decay kernels and their integration values.

References

[1] M. Engelhard, H. Xu, L. Carin, J. A. Oliver, M. Hallyburton, and F. J. McClernon. Predicting smoking events with a time-varying semi-parametric hawkes process model. arXiv preprint arXiv:1809.01740, 2018.
[2] H. Mei and J. M. Eisner. The neural hawkes process: A neurally self-modulating multivariate point process. In Advances in Neural Information Processing Systems, pages 6754–6764, 2017.
[3] Y. Ogata. On lewis’ simulation method for point processes. IEEE Transactions on Information Theory, 27(1):23–31, 1981.
[4] H. Xu, L. Carin, and H. Zha. Learning registered point processes from idiosyncratic observations. In International Conference on Machine Learning, 2018.
[5] H. Xu, X. Chen, and L. Carin. Superposition-assisted stochastic optimization for hawkes processes. arXiv preprint arXiv:1802.04725, 2018.
[6] H. Xu, M. Farajtabar, and H. Zha. Learning granger causality for hawkes processes. In International Conference on Machine Learning, pages 1717–1726, 2016.
Table 5: A list of loss functions.

| Loss Function | Class | Formulation |
|---------------|-------|-------------|
| Maximum Likelihood Estimation | OtherLayers.MaxLogLike | \( L(\theta) = -\sum_{i \in D} \left( \log \lambda_{c_i}(t_i) - \sum_{c \in C} f_{t_{i-1}}^{t_i} \lambda_c(s) ds \right) \) |
| Least Square Estimation | OtherLayers.LeastSquare | \( L(\theta) = \sum_{i \in D} \left\| f_{t_{i-1}}^{t_i} \lambda(s) ds - 1_{c_i} \right\|_2^2 \) |
| Conditional Likelihood Estimation | OtherLayers.CrossEntropy | \( L(\theta) = -\sum_{i \in D} \log p(c_i|t_i, H_i) = -\sum_{i \in D} \log \text{softmax} \left( f_{t_{i-1}}^{t_i} \lambda(s) ds \right) \). |

Figure 22: Illustration the construction of a point process model.

[7] H. Xu, D. Luo, and L. Carin. Online continuous-time tensor factorization based on pairwise interactive point processes. In Proceedings of the 27th International Conference on Artificial Intelligence. AAAI Press, 2018.

[8] H. Xu, D. Luo, X. Chen, and L. Carin. Benefits from superposed hawkes processes. In International Conference on Artificial Intelligence and Statistics, pages 623–631, 2018.

[9] H. Xu, D. Luo, and H. Zha. Learning hawkes processes from short doubly-censored event sequences. In International Conference on Machine Learning, pages 3831–3840, 2017.

[10] H. Xu, W. Wu, S. Nemati, and H. Zha. Patient flow prediction via discriminative learning of mutually-correcting processes. IEEE Transactions on Knowledge and Data Engineering, 29(1):157–171, 2017.

[11] H. Xu, Y. Zhen, and H. Zha. Trailer generation via a point process-based visual attractiveness model. In Proceedings of the 24th International Conference on Artificial Intelligence, pages 2198–2204. AAAI Press, 2015.

[12] Q. Zhao, M. A. Erdogdu, H. Y. He, A. Rajaraman, and J. Leskovec. Seismic: A self-exciting point process model for predicting tweet popularity. In Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pages 1513–1522. ACM, 2015.

[13] K. Zhou, H. Zha, and L. Song. Learning social infectivity in sparse low-rank networks using multi-dimensional hawkes processes. In Artificial Intelligence and Statistics, pages 641–649, 2013.
Table 6: Typical models implemented by PoPy.

| Model                        | Exogenous Intensity                      | Endogenous Impact                      | Decay Kernel          | Activation $g_\lambda$ | Loss               |
|------------------------------|------------------------------------------|----------------------------------------|-----------------------|------------------------|--------------------|
| Linear Hawkes process [13]   | Exogenous Intensity                      | Endogenous Impact                      | Decay Kernel          | Activation $g_\lambda$ | Loss               |
| Linear Hawkes process [6, 5] | LinearExogenousIntensity                | NavieEndogenousImpact                 | ExponentialKernel     | Identity               | MaxLogLike         |
| Linear Hawkes process [8]    | LinearExogenousIntensity                | NavieEndogenousImpact                 | MultiGaussKernel      | Identity               | MaxLogLike         |
| Factorized point process [7] | LinearExogenousIntensity                | FactorizedEndogenousImpact            | ExponentialKernel     | Identity               | LeastSquares       |
| Semi-Parametric Hawkes process [1] | LinearExogenousIntensity                | NavieEndogenousImpact                 | MultiGaussKernel      | Identity               | MaxLogLike         |
| Parametric self-correcting process [11] | LinearExogenousIntensity                | LinearEndogenousImpact                | GateKernel            | Softplus               | MaxLogLike         |
| Mutually-correcting process [10] | LinearExogenousIntensity                | LinearEndogenousImpact                | GaussianKernel        | Softplus               | CrossEntropy       |

17