Steering the Networked Temporal Point Processes via Controlling the Network Graph

Haoyan Luo

School of Data Science
The Chinese University of Hong Kong, Shenzhen
haoyanluo@link.cuhk.edu.cn

Shuang Li

School of Data Science
The Chinese University of Hong Kong, Shenzhen
lishuang@cuhk.edu.cn

Abstract

Multivariate temporal point processes provide an elegant tool to model the discrete events occurring over the networks. How to effectively steer the networked event processes at minimal cost via controlling the network graph, e.g., temporarily delete or create edges? This problem commonly arises when we aim to curb the epidemic by locking down cities or suspending flights, or when we hope to solve the traffic congestion via scheduling the traffic lights. We formulate this problem as a finite-time horizon model-based RL problem where the underlying dynamics of the networked temporal point processes are modeled by neural ODEs. Consider the graph topology, given n nodes, there are n^2 edges and the action space is therefore with size n^2. To speed up the planning and tackle high-dimensionality, we propose to learn neural network policies and pool the optimal policies from history and other regions in decision making. Our proposed policy can naturally incorporate prior knowledge and constraints. We implement our methods on real COVID dataset and achieve promising results.

1 Introduction

We consider a scenario where we aim to steer the dynamics of the multivariate temporal point processes by sequentially modifying the dependency graphs (e.g., add or delete edges) at a minimal cost. The problem is widely motivated by many applications in society. For example, the outbreak of epidemic disease at different places can be modeled as multivariate temporal point processes which are subject to spatial-temporally intertwined dependency structures. The government is responsible to make policies to curb the spread of the disease. An effective intervention approach is to choose to temporarily lockdown some cities or suspend some flights to cut the influence from one place to another place. Although effective, these intervention approaches are usually costly. How to smartly design such a lockdown policy, reduce financial costs, boost social welfare, and make everyone feel fair and happy is a challenging problem.

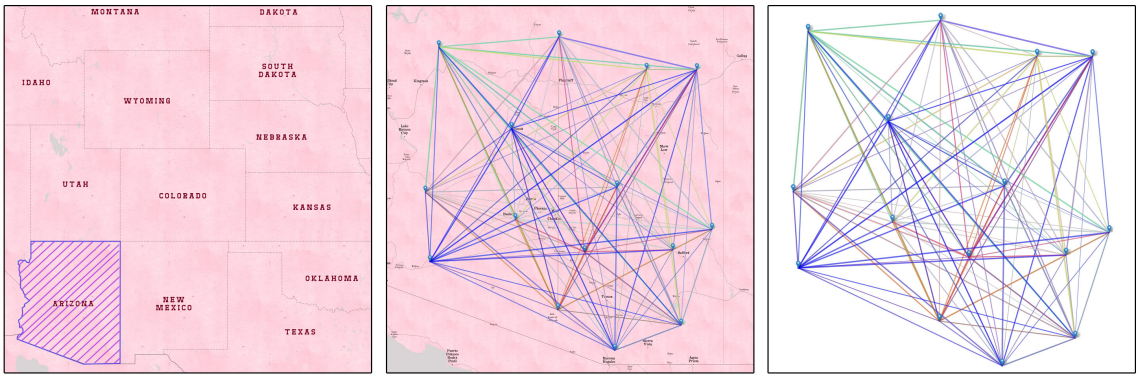
For the traffic flow control, decision makers can execute interventions such as designing traffic light policies to relieve the traffic congestion and realize low carbon. However, some decisions might be difficult to make due to various concerns and constraints. For instance, when making the regulations for controlling the coronavirus, the government needs to consider the effect on the economy and the citizens' attitude towards the regulations. An over-strict regulation may restrict the development of the economy and be complained by the public while a loose one may lead to severe consequence of infections. In this case, a numerical measurement on the influence of those factors needs to be done to

help the government make better decisions. For the traffic light scheduling, one need to accommodate some special needs. For example, we need to consider the period time, peak, flat peak and night, which are respectively focused on capacity, efficiency and safety.
To do the measurements on various factors and help with the decision-making process, we introduce a threshold policy over spatial-temporal process. We focus on the intensity of the spatial-temporal process and learn a parametric threshold model mainly based on the intensity function. The threshold is treated as the boundary of the decision, determining the acceptance or rejection of the policy choices. The threshold policy has many advantages when solving decision-making problems on the spatial-temporal process. It has overall considerations to the influential factors of the decision and reasonably focuses on the intensity of the events. It is easy for the decision maker to adopt and interpret. The threshold model is promising to solve further decision-making problems under spatial-temporal process in different situations.

Our contributions can be summarized as follows:

- **Innovative Control Framework for Networked Events:** Introduction of a model-based reinforcement learning approach using neural ODEs for dynamically steering networked temporal point processes. This framework is particularly applicable in scenarios like epidemic control and traffic flow management, providing a novel method for intervention in complex networked systems.
- **High-Dimensional Policy Learning and Optimization:** Development of efficient strategies for learning and optimizing neural network policies in high-dimensional spaces (size n^2 for n nodes). This includes pooling optimal policies from historical and regional data, enabling effective decision-making in large-scale networks.
- **Threshold Policy for Balanced Decision-Making:** Creation of a parametric threshold model based on intensity functions for spatial-temporal processes. This model facilitates balanced decision-making by considering various influential factors, offering a practical and interpretable tool for policymakers in diverse scenarios.

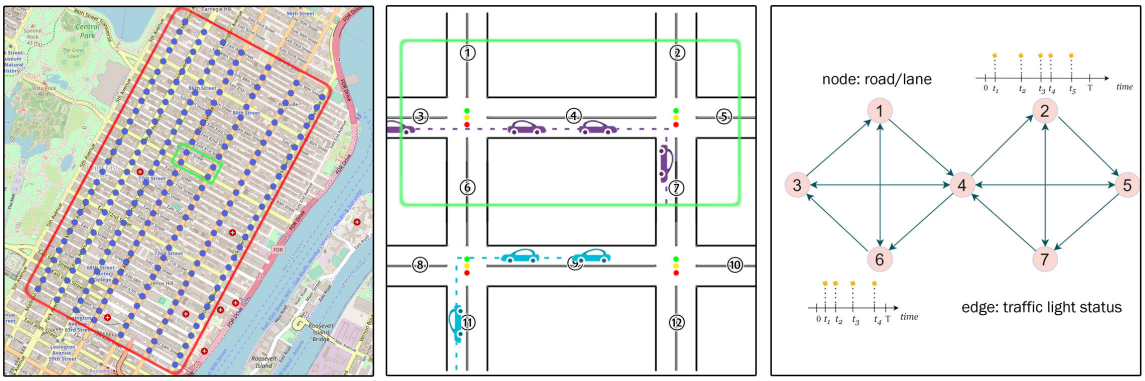
2 Related Work



(a) Part view of USA (b) Cities influence relations in Arizona (c) Influence Graph

Figure 1: An example of Covid-19 spread networks in the US. We studied each state separately, using cities influence relations in Arizona as an example (subfigure (b)). The relationship of cities with serious epidemic situation in the state is based on the graph of subfigure(c).

.



(a) Upper East Side, Manhattan, New York, USA (b) Subregion traffic network (c) Graph of subregion

Figure 2: Subfigure(a) is an example using traffic networks in the upper east side, Manhattan, New York, USA, 196 intersections with uni-directional traffic. Red polygons are the areas we select to model, blue dots are the traffic signals we control. Based on major roads in the traffic network, the entire area is partitioned into regions (subfigure (b)). We use nodes to represent roads/lanes and edges to represent traffic lights. And the roads/lanes traffic influence relations are based on which we create the notion of links (subfigure (c)).

3 Background

3.1 Multivariate Hawkes Process

A temporal point process is a stochastic generative model whose output is a sequence of discrete events $\mathcal{H} = \{t_i\}$. An event sequence can be formally described by a counting function $N(t)$ recording the number of events before time t, which is defined as follows:

$$N(t) = \sum_{t_i \in \mathcal{H}} H\left(t - t_i\right), \quad \text{where} \quad H(t) = \begin{cases} 0 & t \leq 0 \\ 1 & \text{otherwise} \end{cases} \tag{1}$$

Suppose we can split the space into m cells. Modeling the spatio-temporal recurring as a m-dimensional counting process

$$\boldsymbol{N}(t) = \left(N_1(t), N_2(t), \ldots, N_m(t)\right)^\top$$

where $N_i(t)$ records the total number of events generated within the i-th cell up to time t.
Furthermore, we represent each event as a tuple (u_i, t_i), where u_i is the cell identity and t_i is the $i-th$ event timing. Let the history of the process up to time t be

$$\mathcal{H}_t := \{(u_i, t_i) \mid t_i < t\}$$

and $\mathcal{H}_{t-}$ be the history until just before time t.
Then the increment of the process, $d\boldsymbol{N}(t)$, in an infinitesimal window $[t, t + dt]$ is parametrized by the intensity $\boldsymbol{\lambda}(t) = \left(\lambda_1(t), \ldots, \lambda_m(t)\right)^\top$, i.e.,

$$\mathbb{E}\left[d\boldsymbol{N}(t) \mid \mathcal{H}_{t-}\right] = \boldsymbol{\lambda}(t)dt$$

Intuitively, the larger the intensity $\boldsymbol{\lambda}(t)$, the greater the likelihood of observing an event in the time window $[t, t+ dt]$. For instance, a Poisson process in $[0, \infty)$ can be viewed as a special counting process with a constant intensity function λ, independent of time and history.

A multivariate Hawkes process is a counting process who has a particular form of intensity. We assume that the strength of influence between users is parameterized by a sparse nonnegative influence matrix $\boldsymbol{A} = \left(a_{uu'}\right)_{u,u' \in [m]}$, where $\alpha_{uu'} > 0$ means cell u' directly excites cell u. We also allow $\boldsymbol{A}$ to have **nonnegative** diagonals to model self-excitation within a cell. Then, the intensity of the u-th

dimension is

$$\begin{aligned}\lambda_u(t) &= \mu_u + \lambda^*(t) \\ &= \mu_u + \sum_{i:t_i<t} a_{uu'} g\left(t - t_i\right) = \mu_u + \sum_{u' \in [m]} a_{uu'} \int_0^t g(t-s) dN_{u'}(s)\end{aligned} \tag{2}$$

The common star superscript shorthand $\lambda^*(t) = \mathbf{a}_(t|\mathcal{H}_t)$. These events often exhibit the characteristics of self- and mutual-excitation, where the former models that a new event might be triggered by the historical events happening within the same cell recently, and the latter models that a new event might be triggered by the historical events happening in the neighboring cells. This type of multivariate temporal point process is usually used to model the spread of epidemics, e.g., COVID-19.

Impulse input (COVID) We can treat the base intensity μ as the input of the system. For COVID, we can consider μ as a impulse function, i.e., $\mu(t)dt = 1$ at $t = 0$ and $\mu(t) = 0$ for all other times.

Constant input (Traffic) For the traffic flow, we can consider μ as a deterministic function of t, i,e., $\mu(t)$. To make it simple, we model it as a constant.

3.2 Parametric Model: Hawkes process with exponential decay function

A well-known result is that if we assume an exponential decay functions

$$g(t) = \exp(-\beta t)$$

then a nice property is that $(N_u, \mathbf{a}_u)$ is a Markov process, and the intensity function has a Markovian property and the generating process of the intensity is a Markov process.

This property can be extended to the case where the decay function $g(t)$ is a finite sum of exponentials

$$g(t) = \sum_{k=1,\ldots,K} g_k(t) = \sum_{k=1,\ldots,K} \beta_k \exp(-\beta_k t) \tag{3}$$

where $g_k(t) = \beta_k \exp(-\beta_k t)$ and there are K components and $g_k(t) = \beta_k \exp(-\beta_k t)$ or $g_k(t) = \exp(-\beta_k t)$.

4 Our Environment Model

4.1 Neural ODE Model for Multivariate Temporal Point Processes

We take inspiration from the Neural Hawkes Process to model the events over the space and time. Consider a multi-variate temporal point process, where each dimension represents the counting process of a cell. More specifically, we introduce

$$\mathbf{z} = [\mathbf{z}_1, \mathbf{z}_2, \ldots, \mathbf{z}_u, \ldots, \mathbf{z}_m]$$

where each latent vector $\mathbf{z}_u$ is to model the latent state of the u-th dimension,the latent vector $\mathbf{z}_u$ flows continuously over time until an event happens at random, which introduces an abrupt jump and changes its trajectory.

Starting from the initial state $\mathbf{z}_u\left(t_0\right)$, the transformed state at any time t_i is given by integrating an ODE forward in time:

$$\frac{d\mathbf{z}_u(t)}{dt} = f(\mathbf{z}_u(t), t; \theta), \quad \mathbf{z}_u\left(t_i\right) = \mathbf{z}_u\left(t_0\right) + \int_{t_0}^{t_i} \frac{d\mathbf{z}_u(t)}{dt} dt$$

Let $N_u(t)$ be the event number of dimension u up to time t. The latent state dynamics of our model is described by the following equation:

$$d\mathbf{z}_u(t) = \int_0^t f(\mathbf{z}_u(t), t)dt + \sum_{u' \in [m]} A_{u'u}\phi(\mathbf{z}_{u'}(t), t)dN_{u'}(t) \tag{4}$$

where f and ϕ are two neural networks that control the continuous changes and instantaneous jump, respectively.

Following the definition of the counting function (Eq.1), all time dependent variables are left continuous in t, i.e., $\lim_{\epsilon \to 0^+} \mathbf{z}(t-\epsilon) = \mathbf{z}(t)$. And denote the right limit of any time dependent variable $x(t)$ by $x\left(t^+\right) = \lim_{\epsilon \to 0^+} x(t+\epsilon)$.

$$\mathbf{z}_u(t_0) = \mathbf{z}_0^u \tag{5}$$

$$\frac{d\mathbf{z}_u(t)}{dt} = f(t, \mathbf{z}_u(t), \theta_f) \quad \text{(Continuous evolution)} \tag{6}$$

$$\mathbf{z}_u(t_i^+) = \sum_{u' \in [m]} A_{uu'}\phi(t_i, \mathbf{z}_{u'}(t_i), \theta_\phi) \quad \text{(Instantaneous updates)} \tag{7}$$

Given the above model, we can write out the likelihood of the event sequences for dimension u as below:

$$\mathcal{L}_u = \sum_{i=1}^{n^u} \log \lambda_u^*(\mathbf{z}(t_i^u)) - \int_0^T \lambda_u^*(\mathbf{z}(\tau))d\tau \tag{8}$$

For all the multi-variate event sequences up to time T, we define the likelihood function as below:

$$\mathcal{L} = \log p(\mathcal{H}) = \sum_{u=1}^{m} \left(\sum_{i=1}^{n^u} \log \lambda_u^*(\mathbf{z}(t_i^u)) - \int_0^T \lambda_u^*(\mathbf{z}(\tau))d\tau \right) \tag{9}$$

So the goal is to learn all the model parameters so as to maximize the log-likelihood.

4.2 Solve the Neural ODE Multivariate Temporal Point Processes

In practice, the integral in Eq.(8) is computed by a weighted sum of intensities $\lambda\left(\mathbf{z}_u\left(t_i\right)\right)$ on checkpoints $\{t_i\}$. Therefore, computing the loss function $\mathcal{L}_u = \mathcal{L}_u\left(\{\mathbf{z}_u\left(t_i\right)\}; \theta\right)$ requires integrating Eq.(4) forward from t_0 to t_N and recording the latent vectors along the trajectory.

4.2.1 Continuous Dynamics function $f(\mathbf{z}_u)$

Starting from the input layer $\mathbf{z}_0$, we can define the output layer $\mathbf{z}_T$ to be the solution to this ODE initial value problem at some time T. This value can be computed by a black-box differential equation solver, which evaluates the hidden unit dynamics f wherever necessary to determine the solution with the desired accuracy.

Consider optimizing a scalar-valued loss function $\mathcal{L}_u$, whose input is the result of continuous ODE solver:

$$\mathcal{L}_u\left(\mathbf{z}_u\left(t_1\right)\right) = \mathcal{L}_u\left(\mathbf{z}_u\left(t_0\right) + \int_{t_0}^{t_1} f(\mathbf{z}_u(t), t, \theta)dt\right) = \mathcal{L}_u\left(\text{ODESolve-Continuous}\left(\mathbf{z}_u\left(t_0\right), f, t_0, t_1, \theta\right)\right)$$

To optimize the above Loss function $\mathcal{L}_u$, we require gradients with respect to $\mathbf{z}_u(t_0), t_0, t_1, \theta$. The first step is to determining how the gradient of the loss depends on the hidden state $\mathbf{z}_u(t)$ at each

instant.This quantity is called the adjoint state $\mathbf{a}_u(t) = \mathcal{L}_u/\partial \mathbf{z}_u(t)$. Its dynamics are given by another ODE, which can be thought of as the instantaneous analog of the chain rule:

$$\mathbf{a}_u(t) = \frac{\partial \mathcal{L}_u}{\partial \mathbf{z}_u(t)} = \frac{\partial \mathcal{L}_u}{\partial \mathbf{z}_u(t^+)} \frac{\partial \mathbf{z}_u(t^+)}{\partial \mathbf{z}_u(t)} = \mathbf{a}_(t^+) \frac{\partial \mathbf{z}_u(t^+)}{\partial \mathbf{z}_u(t)} \tag{10}$$

Unlike the discrete changes in traditional neural networks, the changes here are infinitesimal. According to the integral solution formula of ODE, given the hidden state $z(t)_u$, we can express $\mathbf{z}_u(t+\epsilon)$ as the integral form:

$$\mathbf{z}_u(t+\epsilon) = \mathbf{z}_u(t) + \int_t^{t+\epsilon} f(\mathbf{z}_u(t), t, \theta) dt \tag{11}$$

Thus we have

$$\begin{aligned} \mathbf{a}_u(t) &= \mathbf{a}_u(t+\epsilon) \frac{\partial}{\partial \mathbf{z}_u(t)} (\mathbf{z}_u(t) + \int_t^{t+\epsilon} f(\mathbf{z}_u(t), t, \theta) dt) \\ &= \mathbf{a}_u(t+\epsilon) + \mathbf{a}_u(t+\epsilon) \frac{\partial}{\partial \mathbf{z}_u(t)} (\int_t^{t+\epsilon} f(\mathbf{z}_u(t), t, \theta) dt) \end{aligned} \tag{12}$$

Combining the above equations, we can derive a differential equation describing the adjoint state change over time:

$$\begin{aligned} \frac{d\mathbf{a}_u(t)}{dt} &= \lim_{\epsilon \to 0^+} \frac{\mathbf{a}_u(t+\epsilon) - \mathbf{a}_u(t)}{\epsilon} \\ &= \lim_{\epsilon \to 0^+} \frac{-\mathbf{a}_u(t+\epsilon) \frac{\partial}{\partial \mathbf{z}_u(t)} (\int_t^{t+\epsilon} f(\mathbf{z}_u(t), t, \theta) dt)}{\epsilon} \\ &= \lim_{\epsilon \to 0^+} \frac{-\mathbf{a}_u(t+\epsilon) \frac{\partial}{\partial \mathbf{z}_u(t)} (\epsilon f(\mathbf{z}_u(t), t, \theta))}{\epsilon} \\ &= \lim_{\epsilon \to 0^+} \frac{-\mathbf{a}_u \epsilon (t+\epsilon) \frac{\partial}{\partial \mathbf{z}_u(t)} (f(\mathbf{z}_u(t), t, \theta))}{\epsilon} \\ &= \lim_{\epsilon \to 0^+} -\mathbf{a}_u(t+\epsilon) \frac{\partial}{\partial \mathbf{z}_u(t)} (f(\mathbf{z}_u(t), t, \theta)) \\ &= -\mathbf{a}_u(t) \frac{\partial f(\mathbf{z}_u(t), t, \theta)}{\partial \mathbf{z}_u(t)} \end{aligned} \tag{13}$$

The adjoint state at time t_N is known because it is actually the gradient of the final hidden state with respect to the loss, define the initial condition of the adjoint variables as follows,

$$\mathbf{a}_u(t_N) = \frac{\partial \mathcal{L}_u}{\partial \mathbf{z}_u(t_N)} \quad \mathbf{a}_\theta^u(t_N) = 0, \quad \mathbf{a}_t(t_N) = \frac{\partial \mathcal{L}_u}{\partial t_N} = \frac{\partial \mathcal{L}_u}{\partial \mathbf{z}_u(t_N)} \frac{\partial \mathbf{z}_u(t_N)}{\partial t_N} = \mathbf{a}_u(t_N) f(\mathbf{z}_u(t_N), t_N, \theta) \tag{14}$$

So we can integrate the adjoint state's differential equation backward to solve

$$\begin{aligned} \mathbf{a}_u(t_0) &= \partial \mathcal{L}_u / \partial \mathbf{z}_u(t_0) \\ \mathbf{a}_\theta^u(t_0) &= \partial \mathcal{L}_u / \partial \theta \\ \mathbf{a}_t^u(t_0) &= \partial \mathcal{L}_u / \partial t_0 \end{aligned} \tag{15}$$

$$\begin{aligned} &\frac{d\mathbf{a}_u(t)}{dt} = -\mathbf{a}_u(t) \frac{\partial f(\mathbf{z}_u(t), t, \theta)}{\partial \mathbf{z}_u(t)}, \quad \mathbf{a}_u(t_0) = \mathbf{a}_u(t_N) + \int_{t_N}^{t_0} \left[\frac{d\mathbf{a}_u(t)}{dt} - \sum_{i \neq N} \delta(t - t_i) \frac{\partial \mathcal{L}}{\partial \mathbf{z}_u(t_i)} \right] dt \\ &\frac{d\mathbf{a}_\theta^u(t)}{dt} = -\mathbf{a}_u(t) \frac{\partial f(\mathbf{z}_u(t), t, \theta)}{\partial \theta}, \quad \mathbf{a}_\theta^u(t_0) = \mathbf{a}_\theta^u(t_N) + \int_{t_N}^{t_0} \frac{d\mathbf{a}_\theta^u(t)}{dt} dt \\ &\frac{d\mathbf{a}_t^u(t)}{dt} = -\mathbf{a}_u(t) \frac{\partial f(\mathbf{z}_u(t), t, \theta)}{\partial t}, \quad \mathbf{a}_t^u(t_0) = \mathbf{a}_t^u(t_N) + \int_{t_N}^{t_0} \left[\frac{d\mathbf{a}_t^u(t)}{dt} - \sum_{i \neq N} \delta(t - t_i) \frac{\partial \mathcal{L}}{\partial t_i} \right] dt \end{aligned} \tag{16}$$

4.2.2 Jump function$\phi(h)$

When an event happens at timestamp t_i in dimension u,we can express $\mathbf{z}_u(t_i^+)$ as below:

$$\mathbf{z}_u(t_i^+) = \mathbf{z}_u(t_i) + A_{u'u}\phi(\mathbf{z}_{u'}(t_i), t_i, \theta) \tag{17}$$

consider optimizing the following loss function $\mathcal{L}_u$:

$$\mathcal{L}_u(\mathbf{z}_u(t_i^+)) = \mathcal{L}_u\left(\mathbf{z}_u(t_i) + A_{u'u}\phi(\mathbf{z}_{u'}(t_i), t_i, \theta)\right) \tag{18}$$

where all the time dependent variables are left continuous in time.

$$\mathbf{a}_u(t_i) = \frac{\partial \mathcal{L}_u}{\partial \mathbf{z}_u(t_i)} = \frac{\partial \mathcal{L}_u}{\partial \mathbf{z}_u(t_i^+)}\frac{\partial \mathbf{z}_u(t_i^+)}{\partial \mathbf{z}_u(t_i)} = \mathbf{a}_u(t_i^+)\frac{\partial \mathbf{z}_u(t_i^+)}{\partial \mathbf{z}_u(t_i)} \tag{19}$$

combine above formulas we get

$$\begin{aligned} a_u(t_i) &= \mathbf{a}_u(t_i^+)\frac{\partial(\mathbf{z}_u(t_i) + A_{u'u}\phi(\mathbf{z}_{u'}(t_i), t_i, \theta)}{\partial \mathbf{z}_u(t_i)} \\ &= \mathbf{a}_u(t_i^+)(\boldsymbol{I} + \frac{\partial A_{u'u}\phi(\mathbf{z}_{u'}(t_i), t_i, \theta)}{\partial \mathbf{z}_u(t_i)}) \\ &= \mathbf{a}_u(t_i^+) + \mathbf{a}_u(t_i^+)\frac{\partial}{\partial \mathbf{z}_u(t_i)}A_{u'u}\phi(\mathbf{z}_{u'}(t_i), t_i, \theta) \end{aligned} \tag{20}$$

similiarly, we have

$$\mathbf{a}_\theta^u(t_i) = \frac{\partial \mathcal{L}_u}{\partial \theta} = \mathbf{a}_\theta^u(t_i^+) + \mathbf{a}_u(t_i^+)\frac{\partial}{\partial \theta}A_{u'u}\phi(\mathbf{z}_{u'}(t_i), t_i, \theta) \tag{21}$$

$$\mathbf{a}_t^u(t_i) = \frac{\partial \mathcal{L}_u}{\partial t_i} = \mathbf{a}_t^u(t_i^+) + \mathbf{a}_u(t_i^+)\frac{\partial}{\partial t_i}A_{u'u}\phi(\mathbf{z}_{u'}(t_i), t_i, \theta) \tag{22}$$

$$\frac{\partial \mathcal{L}_u}{\partial A} = \phi(\mathbf{z}_{u'}(t_i), t_i, \theta) \tag{23}$$

Moreover, (Eq.19) can be generalized to obtain the jump of $\mathbf{a}_\theta^u$ and $\mathbf{a}_t^u$ at the discontinuities. In the work of [Chen et al., 2020], the authors define an augmented latent variables and its dynamics as,

$$\mathbf{z}_{\text{aug}}(t) = \begin{bmatrix} \mathbf{z} \\ \theta \\ t \end{bmatrix}(t), \quad \frac{d\mathbf{z}_{\text{aug}}(t)}{dt} = f_{\text{aug}}(\mathbf{z}, t, \theta) = \begin{bmatrix} f(\mathbf{z}, t, \theta) \\ \mathbf{0} \\ 1 \end{bmatrix}, \quad \mathbf{a}_{\text{aug}}(t) = \begin{bmatrix} \mathbf{a} & \mathbf{a}_\theta & \mathbf{a}_t \end{bmatrix}(t).$$

Following the same convention, we define the augmented jump function at t_i as,

$$\phi_{\text{aug}}^u(\mathbf{z}_u(t_i), t_i, \theta) = \begin{bmatrix} \phi(\mathbf{z}_{u'}(t_i), t_i, \theta) \\ \mathbf{0} \\ 0 \end{bmatrix}.$$

We can verify that the left and right limits of the augmented latent variables satisfy

$$\mathbf{z}_{\text{aug}}^u(t_i^+) = \begin{bmatrix} \mathbf{z}_u(t_i) \\ \theta \\ t_i \end{bmatrix} + \begin{bmatrix} A_{u'u}\phi(\mathbf{z}_{u'}(t_i), t_i, \theta) \\ \mathbf{0} \\ 0 \end{bmatrix} = \mathbf{z}_{\text{aug}}^u(t_i) + A_{u'u}\phi_{\text{aug}}(\mathbf{z}_{u'}(t_i), t_i, \theta).$$

Algorithm 1 Dynamics Simulation

Input: start time t_0, stop time t_N, Influence Matrix A, initial state $\mathbf{z}(t_0)$

$t = t_0$, $\mathbf{z} = \mathbf{z}(t_0)$,$\mathcal{H} = \{\}$
while $t<t_N$ **do**
  $dt = ODE - AdpativeForwardStepSize(\mathbf{z}, t, \theta)$
  $\tau_j = SimulateNextEventsTime(\mathcal{H}_t)$
  $u_j = SimulateEventsDimensions(\tau_j)$
  if $\tau_j > t + dt$ **then**
    $\mathbf{z} = Forward(\mathbf{z}, dt, \theta)$
  else
    $\mathcal{H} = \mathcal{H} \cup \{(\tau_j, u_j)\}$
    $j = j + 1$
    $dt = \tau_j - t$
    $\mathbf{z} = StepForward(\mathbf{z}, dt, \theta)$
    $\mathbf{z} = JumpForward(\mathbf{z}, (\tau_j, u_j), \theta)$
  end if
  $t+ = dt$
end while

Output: event sequence $\mathcal{H}$

The augmented dynamics is a special case of the general Neural ODE framework, and the jump of adjoint variables can be calculated as

$$\mathbf{a}^u_{\text{aug}}(t_i) = \mathbf{a}^u_{\text{aug}}(t_i^+)\left(\frac{\partial \mathbf{z}^u_{\text{aug}}(t_i^+)}{\partial \mathbf{z}^u_{\text{aug}}(t_i)}\right) = \left[\begin{array}{ccc} \mathbf{a}_u & \mathbf{a}^u_\theta & \mathbf{a}^u_t \end{array}\right](t_i^+) \left[\begin{array}{ccc} \mathbf{I} + \frac{\partial \phi}{\partial \mathbf{z}_u(t_i)} & \frac{\partial \phi}{\partial \theta} & \frac{\partial \phi}{\partial t_i} \\ \mathbf{0} & \mathbf{I} & \mathbf{0} \\ 0 & 0 & \mathbf{I} \end{array}\right],$$

which are equivalent to Eq.(20,21,22). And as they are left continuous right limits, we can get $\mathbf{a}_u(t_i), \mathbf{a}^u_\theta(t_i), \mathbf{a}^u_t(t_i)$ from the continuous part.

5 Policy

5.1 Formulation

Our decision-making process is as follows. Let T be the time-window to look ahead. We will make a decision in the beginning of the T time-window. The action will be executed through out this time window. Note that we don't want to switch the lockdown decisions during the time window.

5.1.1 Action

The objective of the control problem is to minimize the cumulative **new** occurred events and meanwhile reduce the cost. To make the problem simple. Instead of considering an RL problem with budget constraints, let's first directly assume the objective has two-fold goals and it is a weighted linear combination of the two goals.

There are m cells, each with historical data $\mathcal{H}_{t,m}$. We model the COVID confirmed cases as a m-dimension multivariate Hawkes process, $(N_u(t), \mathbf{a}_u(t))$, $u = 1, \ldots, m$. There is a given m-vector of the cost $\boldsymbol{c} = [c_u]$. The entry $c_u > 0$ represents the cost occurred for cell u suppose we want to lockdown cell u.

At time t, suppose we choose to lockdown cell u. On the one hand, a cost c_u will occur. But on the other hand, some elements of the non-negative influence matrix $\boldsymbol{A}(t) = \left(a_{uu'(t)}\right)_{u,u' \in [m]}$, will be shut

Algorithm 2 Computing loss & its derivatives

Input: start time t_0, stop time t_N, Influence Matrix A,initial state $\mathbf{z}(t_0)$,event sequence $\mathcal{H}$

$t = t_0$, $\mathbf{z} = \mathbf{z}(t_0)$
while $t<t_N$ **do**
  $dt = ODE - AdpativeForwardStepSize(\mathbf{z}, t, \theta)$
  $\tau_j = GetNextEventTime(\mathcal{H}, t)$
  $u_j = GetNextEventDimension(\tau_j)$
  if $\tau_j>t + dt$ **then**
    $\mathbf{z} = Forward(\mathbf{z}, dt, \theta)$
  else
    $dt = \tau_j - t$
    $\mathbf{z} = StepForward(\mathbf{z}, dt, \theta)$
    $\mathbf{z} = JumpForward(\mathbf{z}, (\tau_j, u_j), \theta)$
  end if
  $t+ = dt$
end while
$\mathcal{L} = \mathcal{L}(\{\mathbf{z}(t_i)\}, \{\mathbf{z}(\tau_j)\}, \theta)$ *# compute loss function*, $i = 0, N$
$t = t_N$, $a = \frac{\partial \mathcal{L}}{\partial z(t_N)}$, $\mathbf{a}_\theta = 0, \mathbf{a}_t = a \cdot f(z(t_N), t_N, \theta), z = z(t_N)$
while $t>t_0$ **do**
  dt = ODE-AdpativeBackStepSize$(z, a, \mathbf{a}_\theta, \mathbf{a}_t, t, \theta)$
  $\tau_j = PreviousEventsTime(\mathcal{H}_t)$
  $u = EventsDimensions(\tau_j)$
  if $\tau_j<t - dt$ **then**
    $\mathbf{z}, a, a_\theta, \mathbf{a}_t = StepBackward(z, a, \mathbf{a}_\theta, \mathbf{a}_t, t, \theta)$
  else
    $dt = t - \tau_j$
    $\mathbf{z}, a, \mathbf{a}_\theta, \mathbf{a}_t = StepBackward(\mathbf{z}, a, \mathbf{a}_\theta, \mathbf{a}_t, t, \theta)$
    $\mathbf{z}, a, \mathbf{a}_\theta, \mathbf{a}_t = JumpBackward(\mathbf{z}, a, \mathbf{a}_\theta, \mathbf{a}_t, t, \theta)$
  end if
  $t- = dt$
end while

Output: event sequence $\mathcal{H}$

down to zero. More specifically,

$$\begin{cases} a_{uu}(\tau) = 0 & \tau \geq t \\ a_{u'u}(\tau) = 0 & \forall u' \neq u, \tau \geq t \end{cases}$$

In other words, $\boldsymbol{A}_{\cdot u} = 0$.

Let $\boldsymbol{y}$ be the controller at decision point t, which is an $m\times1$ Boolean vector denoting whether the lock down policy is conducted or not.

$$y_u(t) = \begin{cases} 1 & \text{if choose to lock down the } u^{th} \text{ cell at time t} \\ 0 & \text{otherwise} \end{cases}$$

Before the lock-down decision time τ, the influence matrix is $\boldsymbol{A}$. After the decision, the influence matrix becomes

$$\boldsymbol{A}(\boldsymbol{y}) = \boldsymbol{A}\,\mathrm{diag}(\mathbf{1} - \boldsymbol{y}) \qquad t > \tau \tag{24}$$

if we only cut the influence from the lockdown place to other places. Or if we also want to cut the influence from other places to the lockdown places, the influence matrix becomes

$$\boldsymbol{A}(\boldsymbol{y}) = \mathrm{diag}(\mathbf{1} - \boldsymbol{y})\boldsymbol{A}\,\mathrm{diag}(\mathbf{1} - \boldsymbol{y}) \qquad t > \tau \tag{25}$$

In this problem, the state s_t is a compact representation of the entire history $\mathcal{H}_t$. For exponential kernel, we may have an explicit representation of this state.

5.1.2 Dynamics

Let us derive the analytical computation of the mean-field approximation of the dynamics

Impulse input Consider the recursive updating expression of $\boldsymbol{\Lambda}(t+\tau)$, suppose we are using the exponential triggering kernel functions.

$$\boldsymbol{\Lambda}(t+\delta) = e^{-\beta\delta}\boldsymbol{\Lambda}(t) + \boldsymbol{A}(\boldsymbol{y})\int_t^{t+\delta} g(t+\delta-\tau)\boldsymbol{\Lambda}(\tau)d\tau \tag{26}$$

Subtracting $\boldsymbol{\Lambda}(t)$ from the left and right sides of (30), and dividing the both sides by δt, we have

$$\frac{\boldsymbol{\Lambda}(t+\delta)-\boldsymbol{\Lambda}(t)}{\delta} = \frac{e^{-\beta\delta}-1}{\delta}\boldsymbol{\Lambda}(t) + \boldsymbol{A}(\boldsymbol{y})\frac{\int_t^{t+\delta} g(t+\delta-\tau)\boldsymbol{\Lambda}(\tau)d\tau}{\delta}. \tag{27}$$

Taking $\delta \to 0$, it further yields that

$$\dot{\boldsymbol{\Lambda}}(t) = -\beta\boldsymbol{\Lambda}(t) + g(0)\boldsymbol{A}(\boldsymbol{y})\boldsymbol{\Lambda}(t) = (\boldsymbol{A}(\boldsymbol{y}) - \beta I)\boldsymbol{\Lambda}(t)$$

For the first-order differential equation in the mean-field intensity, we have the following solution

$$\boldsymbol{\Lambda}(t) = e^{(\boldsymbol{A}(\boldsymbol{y})-\beta\boldsymbol{I})t} \tag{28}$$

and the recursive form

$$\boldsymbol{\Lambda}(t+\delta) = e^{(\boldsymbol{A}(\boldsymbol{y})-\beta\boldsymbol{I})\delta}\boldsymbol{\Lambda}(t) \tag{29}$$

Constant input

$$\boldsymbol{\Lambda}(t+\delta) = \boldsymbol{\mu} + e^{-\beta\delta}[\boldsymbol{\Lambda}(t)-\boldsymbol{\mu}] + \boldsymbol{A}(\boldsymbol{y})\int_t^{t+\delta} g(t+\delta-\tau)\boldsymbol{\Lambda}(\tau)d\tau \tag{30}$$

Let $\tilde{\mathbf{a}}_(t) = \mathbf{a}_(t) - \boldsymbol{\mu}$. Then

$$\tilde{\boldsymbol{\Lambda}}(t+\delta) = e^{-\beta\delta}\tilde{\boldsymbol{\Lambda}}(t) + \boldsymbol{A}(\boldsymbol{y})\int_t^{t+\delta} g(t+\delta-\tau)\boldsymbol{\Lambda}(\tau)d\tau$$

We invoke (27) and have

$$\dot{\tilde{\boldsymbol{\Lambda}}}(t) = -\beta\tilde{\boldsymbol{\Lambda}}(t) + g(0)\boldsymbol{A}(y)\boldsymbol{\Lambda}(t).$$

Since $\dot{\tilde{\boldsymbol{\Lambda}}}(t) = \dot{\boldsymbol{\Lambda}}$,

$$\dot{\boldsymbol{\Lambda}}(t) = (\boldsymbol{A}(y) - \beta I)\boldsymbol{\Lambda}(t) + \beta\boldsymbol{\mu}.$$

We will have the following solution

$$\boldsymbol{\Lambda}(t) = e^{(\boldsymbol{A}(\boldsymbol{y})-\beta\boldsymbol{I})t}\boldsymbol{\Lambda}(0) + \beta\int_0^t e^{(\boldsymbol{A}(y)-\beta\boldsymbol{I})(t-\tau)}d\tau\boldsymbol{\mu}$$

When $\boldsymbol{A}(y) - \beta\boldsymbol{I}$ is invertible, the second term of the above formula can be further simplified, further yielding that

$$\boldsymbol{\Lambda}(t) = e^{(\boldsymbol{A}(\boldsymbol{y})-\beta\boldsymbol{I})t}\boldsymbol{\Lambda}(0) + \beta(\boldsymbol{A}(y)-\beta\boldsymbol{I})^{-1}(e^{(\boldsymbol{A}(y)-\beta\boldsymbol{I})t} - \boldsymbol{I})\boldsymbol{\mu}.$$

5.1.3 Objective

From the previous derivations, we can define the current state as the expected intensity by conditioning on the history.

$$\boldsymbol{s}_i := \boldsymbol{\Lambda}(t+i\Delta) = \mathbb{E}[\boldsymbol{\lambda}(t+i\Delta)|\mathcal{H}_t], \quad i = 0, \dots, n$$

Consider a finite horizon optimization problem

$$C_{\boldsymbol{y}_{0:n}} = \mathbb{E}\left[\sum_{u\in[m]}\int_t^{t+n\Delta} dN_u(\tau) + \int_t^{t+n\Delta} \boldsymbol{y}(\tau)^\top \boldsymbol{c} d\tau \middle| \mathcal{H}_t\right] \tag{31}$$

$$= \underbrace{\mathbb{E}\left[\sum_{u\in[m]}\int_t^{t+\Delta t} dN_u(\tau) + \int_t^{t+\Delta t} \boldsymbol{y}(\tau)^\top \boldsymbol{c} d\tau \middle| \mathcal{H}_t\right]}_{\text{one-step cost}} + \underbrace{\mathbb{E}\left[\sum_{u\in[m]}\int_{t+\Delta t}^{t+n\Delta} dN_u(\tau) + \int_{t+\Delta t}^{t+n\Delta} \boldsymbol{y}(\tau)^\top \boldsymbol{c} d\tau \middle| \mathcal{H}_t\right]}_{\text{cost to go}} \tag{32}$$

Check the first term

$$\text{one-step cost} = \mathbb{E}\left[\sum_{u\in[m]}\int_t^{t+\Delta t} dN_u(\tau) + \int_t^{t+\Delta t} \boldsymbol{y}(\tau)^\top \boldsymbol{c} d\tau \middle| \mathcal{H}_t\right] \tag{33}$$

$$= \mathbb{E}\left[\sum_{u\in[m]}\int_t^{t+\Delta t} \mathbf{a}_u(\tau) d\tau + \int_t^{t+\Delta t} \boldsymbol{y}(\tau)^\top \boldsymbol{c} d\tau \middle| \mathcal{H}_t\right] \tag{34}$$

$$= \sum_{u\in[m]}\int_t^{t+\Delta t} \bar{\lambda}_u(\tau) d\tau + \int_t^{t+\Delta t} \boldsymbol{y}(\tau)^\top \boldsymbol{c} d\tau \tag{35}$$

$$= \underbrace{\mathbf{1}^\top \int_t^{t+\Delta t} e^{(\boldsymbol{A}(\boldsymbol{y}_0)-\beta \boldsymbol{I})\tau} d\tau \boldsymbol{\lambda}(t) + \Delta t \boldsymbol{y}_0^\top \boldsymbol{c}}_{c_1(\boldsymbol{\Lambda}(0), \boldsymbol{y}_0)} \tag{36}$$

Formulate the problem as a multi-stage optimization. We have

$$\min_{\boldsymbol{y}_{0:n}} \quad \sum_{i=0}^{n} c_i(\boldsymbol{\Lambda}(i), \boldsymbol{y}_i) \tag{37}$$

$$s.t. \quad \boldsymbol{\Lambda}(i+1) = f(\boldsymbol{\Lambda}(i), \boldsymbol{y}_i) \qquad i = 0, \dots, n \tag{38}$$

6 Algorithm

It is challenging the solve the above optimization problem due to that the action space is huge.

6.1 PPO Method

We aim to train a neural-based policy that can sequentially generate the list of planned deleted edges so as to minimize the cumulative cost over time . We aim to learn the model parameters of the neural-based policy in an end-to-end fashion. (Note that there might be others). We propose to use a neural based search algorithm to search for the optimal trajectory.

6.1.1 Policy Gradient and Policy Pooling

Motivation of policy pooling For each dynamic $\boldsymbol{A}$ (regarded as a task), we aim to learn an optimal policy that can steer the point process flow as we want. However, it is super inefficient to learn the optimal policy for each $\boldsymbol{A}$ separately, since intuitively similar dynamics $\boldsymbol{A}$ may lead to similar optimal policies. So we propose to learn a meta-policy in the offline planning stage. In the online policy learning stage, by interacting with with real environments, we will continue to update this meta-policy

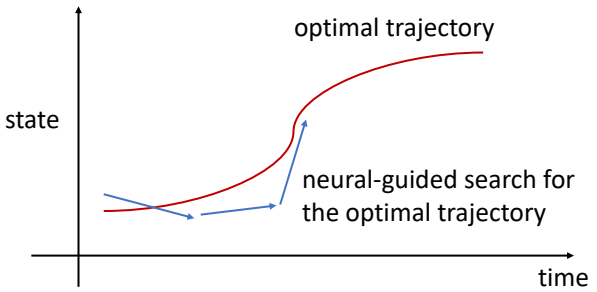


Figure 3: Illustration of the neural-guided search for the optimal trajectory.

by conditional on the dynamic $\boldsymbol{A}$.

The proposed framework is as follows. Given n tasks, each with an estimated $\boldsymbol{A}_i$. We denote $\mathbf{a}_t \circ \boldsymbol{A}_i$ as an operation that forces the corresponding elements of $\boldsymbol{A}_i$ as zeros.

Define an encoder f_ϕ that maps the input graph $\boldsymbol{A}$ to some *low-dimensional* feature space. We formulate meta-offline planning problem as follows

Meta-offline planning

$$\pi_{\theta^*}, f_{\phi^*} = \underset{\pi_\theta, f_\phi}{\operatorname{argmin}} \sum_{i=1}^{n} \mathbb{E}_{\pi_\theta} \left[\sum_{t=1}^{H} c(\mathcal{H}(t), \mathbf{a}_t, f_\phi(\mathbf{a}_t \circ \boldsymbol{A}_i)) \right] \tag{39}$$

In the online policy learning step, we will use π_{θ^*} as the initial policy and freeze f_{ϕ^*}. We will continued to update policy using the real feedback from the environment (feedback generated by simulator). While in the COVID case, the environment has been modelled into A matrices.
Our objective is to find a policy θ that creates a state-action trajectory ξ,

$$\xi = (\boldsymbol{s}_1, \boldsymbol{y}_1, \boldsymbol{s}_2, \boldsymbol{y}_2, ..., \boldsymbol{s}_n, \boldsymbol{y}_n)$$

that minimizes the expected cumulative costs above.
We aim to parameterize the policy as a deep neural network. For example, we introduce the LSTM to model the conditional probability of the edge deletion at time t_i, $i = 0, 1, \ldots, n$,

$$\pi(\boldsymbol{y}_i | \boldsymbol{s}_i)$$

In our problem, the action $\boldsymbol{y}$ is an $m \times 1$ Boolean vector denoting whether the turn on or turn off the edge, the current state is computed by conditioning on the history, $\boldsymbol{s}_\tau := \bar{\boldsymbol{\Lambda}}(\tau) = \mathbb{E}[\boldsymbol{\Lambda}(\tau) | \mathcal{H}_t]$, $\tau \geq t$. For each edge, the probability of deletion can be parameterized as a Bernoulli distribution.

We can train the LSTM model using policy gradient. The training in policy gradient makes actions with high rewards more likely, or vice versa. In reinforcement learning, the instinct may be mathematically described as the probability of taking the action y given a state
To update the policy θ so that it can optimize our objective function, we can write the policy gradient as:

$$\begin{aligned}
\nabla_\theta J(\theta) &= \nabla_\theta (\mathbb{E}_{\xi \sim \pi_\theta(\xi)}[R(\xi)]) \\
&= \nabla_\theta (\int \pi_\theta(\xi) R(\xi) d\xi) \\
&= \int \nabla_\theta \pi_\theta(\xi) R(\xi) d\xi \\
&= \int \pi_\theta(\xi) \nabla_\theta \log \pi_\theta(\xi) R(\xi) d\xi \\
&= \mathbb{E}_{\xi \sim \pi_\theta(\xi)}[\nabla_\theta \log \pi_\theta(\xi) R(\xi)]
\end{aligned} \tag{40}$$

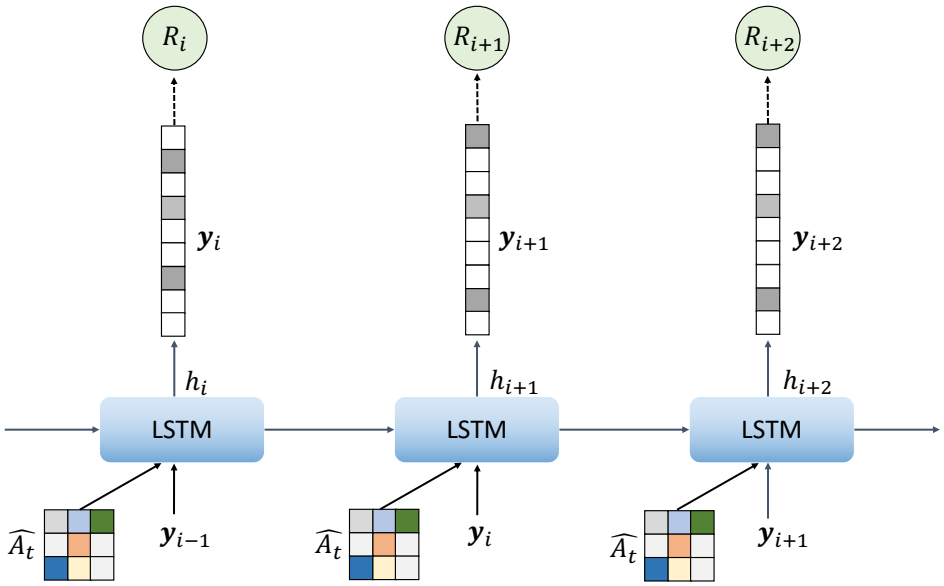


Figure 4: Illustration of the neural-guided search for the optimal trajectory.

Here we use a common trick: $\nabla_\theta \pi_\theta(\xi) = \pi_\theta(\xi) \nabla_\theta \log \pi_\theta(\xi)$. Given the above equation, the policy gradient can be represented as an expectation. It means we can use sampling to approximate it:

$$\nabla_\theta J(\theta) \approx \frac{1}{m} \sum_{k=1}^{m} \sum_{i=1}^{n} \nabla_\theta \log \pi_\theta(y_i^{(k)} | s_i^{(k)}) R_k(\xi) \tag{41}$$

Where m is the total number of the space cells, n is the total number of lock down decisions (actions). Then we use this policy gradient to update the policy θ where α is the learning rate.

$$\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta) \tag{42}$$

6.2 Reparameterization trick

In Appendix A, an approximate problem with analytical solutions is formluated. We will use its analytical solution as an initial guess for our policy. Then we will use policy gradient with gumbel trick to imporve this solution.
(check the feasibility here) Let z be a categorical variable with class probabilities π_1, π_2, ...π_m. The Gumbel-Max trick provides a simple and efficient way to draw samples z from a categorical distribution with class probabilities π

$$z_i = \left\{ \begin{matrix} 1, \, i = \underset{i}{\text{argmax}}(\log(\pi_i) + g_i) \\ 0, \, \text{otherwise} \end{matrix} \right\}, \text{ for } i = 1, ..., m$$

where $u_i \sim \text{Uniform}(0, 1)$ and $g_i = -\log(-\log(u_i))$, which is called the Gumbel random perturbation and is used to make the return value of z_i not fixed. Then we use the softmax function as a continuous, differentiable approximation, and generate m-dimensional controller vector $\boldsymbol{y}$, where

$$y_i = \frac{\exp\left\{(\log(\pi_i) + g_i)/\sigma\right\}}{\sum_{j=1}^{M} \exp\left\{(\log(\pi_j) + g_j)/\sigma\right\}}$$

where σ is the softmax temperature.

The density of the Gumbel-Softmax distribution is

$$p_{\pi,\sigma}(y_1, \ldots, y_m) = \Gamma(m) \sigma^{m-1} \left(\sum_{i=1}^{m} \pi_i / y_i^\sigma \right)^{-m} \prod_{i=1}^{m} \left(\pi_i / y_i^{\sigma+1} \right)$$

Since $\boldsymbol{y} \in \{0,1\}^m$. We assume each element $y_i \sim \text{Bernoulli}(p_i)$ where $p_i \in [0,1]$. Observe that

$$y_i = \frac{\exp\{(\log(p_i) + g_i)/\sigma\}}{\exp\{(\log(p_i) + g_i)/\sigma\} + \exp\{(\log(1-p_i) + g_i')/\sigma\}} \tag{43}$$

This is the Gumbel trick. Also note that in the paper's experiment, the gumbel trick performs almost the same as the slope-annealed straight-through (slope ST) estimator, which should be easier to execute in your experiments. You can use this slope ST estimator for reparametrization. The reference is: Hierarchical multiscale recurrent neural networks. by Bengio ICLR 2017.

The main idea is suppose $y_i \sim \text{Bernoulli}(p_i)$ where $p_i \in [0,1]$, you can use the sigmoid function with slope to approximate the discrete variable y_i (note that the above equation can also be written as a sigmoid function.)

$$y_i = \frac{1}{1+\exp(-\lambda x_i)}, \quad x_i \in R$$

where λ controls the slope. Increasing λ will increase the slope of the sigmoid. The main idea of the slope annealing trick is you should gradually increase the slope of the sigmoid function. So in your training, you should start with a small λ, say 0.001, then gradually increase the value of λ in training.

For your policy, suppose you model it as LSTM. Suppose your current hidden state of LSTM is $\boldsymbol{h} \in R^h$. To obtain $\boldsymbol{y} \in \{0,1\}^m$, you compute

$$\hat{\boldsymbol{y}} = \text{Sigmoid}_\lambda(\boldsymbol{W}\boldsymbol{h}) \tag{44}$$

where $\boldsymbol{W} \in R^{m \times h}$ and λ is tuned from a small number to a big number in the training process.

6.3 Estimate model parameters

First of all, we need to use historical data to estimate the influence matrix $\boldsymbol{A}(t)$. Then We can derive the likelihood function for our multivariate Hawkes process and use maximum likelihood estimation to determine the values of model parameters. The parameter values are found such that they maximise the likelihood produced by the data that were actually observed.

We model the COVID confirmed cases as a m-dimension multivariate Hawkes process, $(N_u(t), \mathbf{a}_u(t)), u = 1, ..., m$ on the time interval $[0, T]$, such that $0 < t_1 < \cdots < t_n \leq T$. Furthermore, we represent each event as a tuple (u_i, t_i), where u_i is the cell identity and t_i is the event timing. Let the history of the process up to time t be

$$\mathcal{H}_t := \{(u_i, t_i) \mid t_i < t\}$$

Denote the set of parameters as θ and given a particular realization ω that contains all points in each dimension $(n_i^u, t_i^u)_{i=1,2,...}$ for $u = 1, 2, ..., m$ on the interval $[0, T]$, the log-likelihood function for our m-dimensional Hawkes process is

$$\ln L\left(\theta \mid \{t_i^u\}_{u=1,2,...,m}\right) = \sum_{u=1}^{m} \ln L^u\left(\theta \mid \{t_i^n\}_{n=1,2,...,m}\right)$$

where

$$\begin{aligned}
\ln L^m\left(\theta \mid \{t_i^n\}_{n=1,2,...,m}\right) &= -\int_0^T \lambda_\theta^u(t \mid \omega)dt + \int_0^T \ln \lambda_\theta^u(t \mid \omega) dN^u(t) \\
&= -\mu_u T - \sum_{n=1}^{m} \frac{\alpha_{un}}{\beta_{un}} \sum_{\{i:t_i^n < T\}} \left[1 - e^{-\beta_{un}(T - t_i^n)}\right] + \sum_{\{i:t_i^u < T\}} \ln\left[\mu_u + \sum_{n=1}^{m} \alpha_{un} R_{un}(i)\right]
\end{aligned}$$

with $R_{un}(i)$ defined recursively as

$$R_{un}(i) = e^{-\beta_{un}\left(t_k^u - t_{k-1}^u\right)} R_{un}(i-1) + \sum_{\{j:t_{i-1}^u \leq t_j^n < t_i^u\}} e^{-\beta_{un}\left(t_i^u - t_j^n\right)}$$

with initial condition:

$$R_{un}(0) = 0$$

7 Baseline Policy (A Naive Policy)

Let's only focus one cell (i.e., individual agent). Each agent can choose to lockdown or lift the lockdown. Individual can get access to the global state. In some sense, we convert the problem to a multi-agent problem.

For the u-th agent, the action is $y_u \in \{0, 1\}^n$. The objective function is

$$C_{y_u}(\boldsymbol{s}_t) = \mathbb{E}\left[\int_t^{t+T} dN_u(\tau) + \int_t^{t+T} y_u(\tau) c_u d\tau \middle| \mathcal{H}_t\right] \tag{45}$$

Let's assume that after the lockdown, the mutual exciting influence will be cut off and only the self-exciting influence may exist.

At t, if no lockdown, $y_t = 0$

$$\lambda_u^{y=0}(t + \Delta t) = e^{-\beta_k \Delta t} \cdot \mathbf{a}_u(t) + \sum_{u' \in [m]} a_{uu'} \int_t^{t+\Delta t} g(t + \Delta t - \tau) dN_{u'}(\tau) \tag{46}$$

At t, if there is a lockdown, $y_t = 1$

$$\lambda_u^{y=1}(t + \Delta t) = e^{-\beta_k \Delta t} \cdot \mathbf{a}_u(t) + a_{uu} \int_t^{t+\Delta t} g(t + \Delta t - \tau) dN_u(\tau) \tag{47}$$

Next, please derive the dynamics as Eq. (**??**) and (**??**). This type of analytical analysis provides us a way to efficiently evaluate the objective function (45) given any policy from the defined policy space.

One can simply perform the exhaustive search. For each agent, computes the optimal policy. Compare this naive policy with our proposed policy.

8 Experiments

8.1 Datasets

Covid-19
We use data released publicly by NY-Times [2020] on daily COVID-19 cases, from June to Dec of 2021. And we want to study the impact between different counties within a certain range or certain neighbors, for example, no more than 300km, 15 nearest neighbors,etc. In order to overcome the difference in the magnitude of the number of patients in different counties and the fact that there were new cases in each city every day in the later period of the data, we treat the data per N(default 500) cases as one event.

Transportation
Li [2018] provides 27 traffic datasets which includes both road network and traffic flow file. For our project, we need to get the congestion data from the raw. According to the data, we extract three kinds of congestion events. One scenario is no vehicle passes through the lane where the traffic light is located during the whole green period, and the number of vehicles in the lane is not 0 when the green light is on. Another situation is that some traffic lights are always green, then anytime there is a waiting vehicle in the lane where the traffic light is located, it is traffic congestion. And the last case is the continuation of congestion, it is possible that the traffic lights turn red and then green, but still no car moved, in this case we need to merge the continuous congestion event into one.

9 Related Work

Jia and Benson [2019] proposed Neural Jump SDEs which incorporates discrete events that abruptly change the latent vector, while this model mainly focus on one dimension with multiple features, not considering the influence between multi-dimensions. Chen et al. [2020] proposed a new class of parameterizations for spatio-temporal point processes with Neural ODEs, however, in their datasets for covid-19, they treat all new cases of data as an event, in other words, no matter how many new cases increase today, they treat it as one event. But different county has different population, the magnitude of the patients increase vary greatly. Besides, with the development of the Covid-19, after the second half of 2020, almost all 3233 counties have new cases every day, even the authors adding the random noise in the range [0,1] to the time sequence data, but the data still changes regularly which does not reflect the magnitude of the data reasonably. (one of the concerns is that the data may be out-dated?)

A Optimal solution of linear quadratic control

First of all, we focus on the optimal control problem in linear settings. Consider a system with dynamic process,

$$\boldsymbol{x}_{t+1} = \boldsymbol{A}_t \boldsymbol{x}_t + \boldsymbol{B}_t \boldsymbol{u}_t, \tag{48}$$

where $t \in [0, n-1]$ is the timestep and n is a finite time horizon, $\boldsymbol{x}$ is the state, $\boldsymbol{u}$ is the control input, and $\boldsymbol{A}$, $\boldsymbol{B}$ are some real matrices(may be time-varying) with feasible size. The initial state $\boldsymbol{x}_0$ is known, and The objective is to find a control sequence $\{\boldsymbol{u}_t \mid t \in [0, n-1]\}$ that minimizes the cost function J,

$$J = \boldsymbol{x}_n^\top \boldsymbol{Q}_n \boldsymbol{x}_n + \sum_{t=0}^{n-1} \left(\boldsymbol{x}_t^\top \boldsymbol{Q}_t \boldsymbol{x}_t + \boldsymbol{u}_t^\top \boldsymbol{R}_t \boldsymbol{u}_t \right), \tag{49}$$

where $\boldsymbol{Q}$ and $\boldsymbol{R}$ are two weighted matrices, $\boldsymbol{x}_n$ is the final state.

Lemma 1 *The problem with system (48) and cost J (49) is called linear quadratic regulator(LQR). The optimal solution is*

$$\boldsymbol{u}_t = -\boldsymbol{K}_t \boldsymbol{x}_t, \tag{50}$$

where

$$\boldsymbol{K}_t = \left(\boldsymbol{R}_t + \boldsymbol{B}_t^\top \boldsymbol{P}_{t+1} \boldsymbol{B}_t \right)^{-1} \boldsymbol{B}_t^\top \boldsymbol{P}_{t+1} \boldsymbol{A}_t, \tag{51}$$

and the nonnegative definite matrix $\boldsymbol{P}_t$ satisfies

$$\boldsymbol{P}_t = \left(\boldsymbol{A}_t - \boldsymbol{B}_t \boldsymbol{K}_t \right)^\top \boldsymbol{P}_{t+1} \left(\boldsymbol{A}_t - \boldsymbol{B}_t \boldsymbol{K}_t \right) + \boldsymbol{K}_t^\top \boldsymbol{R}_t \boldsymbol{K}_t + \boldsymbol{Q}_t, \tag{52}$$

with terminal condition $\boldsymbol{P}_n = \boldsymbol{Q}_n$.

A.1 Generalized result in the nonlinear case

Consider a system with nonlinear dynamic and generalized cost function J,

$$\boldsymbol{x}_{t+1} = f(\boldsymbol{x}_t, \boldsymbol{u}_t), \; J = \sum_{t=0}^{n} g(\boldsymbol{x}_t, \boldsymbol{u}_t) \tag{53}$$

where f and g are two continuous and differentiable functions. The initial state is $\boldsymbol{x}_0$, total time horizon n is finite. The general form of the optimal solution is difficult to solve due to the nonlinearity. Thus, the objective in nonlinear case is to track a predesigned reference trajectory $(\hat{\boldsymbol{x}}_t, \hat{\boldsymbol{u}}_t)$, where the latter $\hat{\boldsymbol{u}}_t$ is not necessary and can be set to 0, it is just used to derive the approximation. Overall,

it would be better if we can set a suitable $\hat{\boldsymbol{u}}$ because the rest is based on the linearization. Given a reference trajectory of $(\hat{\boldsymbol{x}}_t, \hat{\boldsymbol{u}}_t)$, we can linearize it based on the Taylor expansion,

$$\boldsymbol{x}_{t+1} = f(\boldsymbol{x}_t, \boldsymbol{u}_t) \approx f(\hat{\boldsymbol{x}}_t, \hat{\boldsymbol{u}}_t) + J_f(\hat{\boldsymbol{x}}_t, \hat{\boldsymbol{u}}_t) \begin{bmatrix} \boldsymbol{x}_t - \hat{\boldsymbol{x}}_t \\ \boldsymbol{u}_t - \hat{\boldsymbol{u}}_t \end{bmatrix} \tag{54}$$

$$g(\boldsymbol{x}_t, \boldsymbol{u}_t) \approx g(\hat{\boldsymbol{x}}_t, \hat{\boldsymbol{u}}_t) + \begin{bmatrix} \boldsymbol{x}_t - \hat{\boldsymbol{x}}_t \\ \boldsymbol{u}_t - \hat{\boldsymbol{u}}_t \end{bmatrix}^\top J_g(\hat{\boldsymbol{x}}_t, \hat{\boldsymbol{u}}_t) + \begin{bmatrix} \boldsymbol{x}_t - \hat{\boldsymbol{x}}_t \\ \boldsymbol{u}_t - \hat{\boldsymbol{u}}_t \end{bmatrix}^\top H_g(\hat{\boldsymbol{x}}_t, \hat{\boldsymbol{u}}_t) \begin{bmatrix} \boldsymbol{x}_t - \hat{\boldsymbol{x}}_t \\ \boldsymbol{u}_t - \hat{\boldsymbol{u}}_t \end{bmatrix} \tag{55}$$

where J_f is the Jacobi matrix of f, J_g is the Jacobi matrix of g and H_g is the Hessian matrix of g. Then define errors $\delta\boldsymbol{x}_t = \boldsymbol{x}_t - \hat{\boldsymbol{x}}_t$ and $\delta\boldsymbol{u}_t = \boldsymbol{u}_t - \hat{\boldsymbol{u}}_t$, we get the new system model and cost function.

$$\begin{cases} \bar{f}(\delta\boldsymbol{x}_t, \delta\boldsymbol{u}_t) = J_f(\hat{\boldsymbol{x}}_t, \hat{\boldsymbol{u}}_t) \begin{bmatrix} \boldsymbol{x}_t - \hat{\boldsymbol{x}}_t \\ \boldsymbol{u}_t - \hat{\boldsymbol{u}}_t \end{bmatrix} \\ \bar{c}(\delta\boldsymbol{x}_t, \delta\boldsymbol{u}_t) = \begin{bmatrix} \boldsymbol{x}_t - \hat{\boldsymbol{x}}_t \\ \boldsymbol{u}_t - \hat{\boldsymbol{u}}_t \end{bmatrix}^\top J_g(\hat{\boldsymbol{x}}_t, \hat{\boldsymbol{u}}_t) + \begin{bmatrix} \boldsymbol{x}_t - \hat{\boldsymbol{x}}_t \\ \boldsymbol{u}_t - \hat{\boldsymbol{u}}_t \end{bmatrix}^\top H_g(\hat{\boldsymbol{x}}_t, \hat{\boldsymbol{u}}_t) \begin{bmatrix} \boldsymbol{x}_t - \hat{\boldsymbol{x}}_t \\ \boldsymbol{u}_t - \hat{\boldsymbol{u}}_t \end{bmatrix} \end{cases} \tag{56}$$

$$\Rightarrow \begin{cases} \delta\boldsymbol{x}_{t+1} = \boldsymbol{A}_t \delta\boldsymbol{x}_t + \boldsymbol{B}_t \delta\boldsymbol{u}_t \\ \bar{c}_t = \begin{bmatrix} \delta\boldsymbol{x}_t \\ \delta\boldsymbol{u}_t \\ \boldsymbol{1} \end{bmatrix}^\top \begin{bmatrix} \boldsymbol{Q}_{\boldsymbol{x}_t,\boldsymbol{x}_t} & \boldsymbol{Q}_{\boldsymbol{x}_t,\boldsymbol{u}_t} & \boldsymbol{P}_{\boldsymbol{x}_t}^\top \\ \boldsymbol{Q}_{\boldsymbol{u}_t,\boldsymbol{x}_t} & \boldsymbol{Q}_{\boldsymbol{u}_t,\boldsymbol{u}_t} & \boldsymbol{P}_{\boldsymbol{u}_t}^\top \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} \end{bmatrix} \begin{bmatrix} \delta\boldsymbol{x}_t \\ \delta\boldsymbol{u}_t \\ \boldsymbol{1} \end{bmatrix} \end{cases}, \tag{57}$$

where

$$J_f(\hat{\boldsymbol{x}}_t, \hat{\boldsymbol{u}}_t) = \begin{bmatrix} \boldsymbol{A}_t & \boldsymbol{B}_t \end{bmatrix}, \; J_g(\hat{\boldsymbol{x}}_t, \hat{\boldsymbol{u}}_t) = \begin{bmatrix} \boldsymbol{P}_{\boldsymbol{x}_t} & \boldsymbol{P}_{\boldsymbol{u}_t} \end{bmatrix}^\top, \; H_g(\hat{\boldsymbol{x}}_t, \hat{\boldsymbol{u}}_t) = \begin{bmatrix} \boldsymbol{Q}_{\boldsymbol{x}_t,\boldsymbol{x}_t} & \boldsymbol{Q}_{\boldsymbol{x}_t,\boldsymbol{u}_t} \\ \boldsymbol{Q}_{\boldsymbol{u}_t,\boldsymbol{x}_t} & \boldsymbol{Q}_{\boldsymbol{u}_t,\boldsymbol{u}_t} \end{bmatrix}. \tag{58}$$

Finally, the problem comes back to linear form and has a similar solution as Lemma 1, though it may be more complex.

Lemma 2 *For the optimal problem with nonlinear dynamic and generalized cost function, the optimal solution is*

$$\boldsymbol{u}_t = \hat{\boldsymbol{u}}_t + \boldsymbol{K}_t(\boldsymbol{x}_t - \hat{\boldsymbol{x}}_t) + \boldsymbol{v}_t, \tag{59}$$

where

$$\begin{cases} \boldsymbol{K}_t = -\bar{\boldsymbol{Q}}_{\boldsymbol{u}_t,\boldsymbol{u}_t}^{-1} \bar{\boldsymbol{Q}}_{\boldsymbol{u}_t,\boldsymbol{x}_t} \\ \boldsymbol{v}_t = -\frac{1}{2} \bar{\boldsymbol{Q}}_{\boldsymbol{u}_t,\boldsymbol{u}_t}^{-1} \bar{\boldsymbol{P}}_{\boldsymbol{u}_t} \end{cases}, \tag{60}$$

with iterations

$$\begin{cases} \begin{bmatrix} \bar{\boldsymbol{Q}}_{\boldsymbol{x}_t,\boldsymbol{x}_t} & \bar{\boldsymbol{Q}}_{\boldsymbol{x}_t,\boldsymbol{u}_t} \\ \bar{\boldsymbol{Q}}_{\boldsymbol{u}_t,\boldsymbol{x}_t} & \bar{\boldsymbol{Q}}_{\boldsymbol{u}_t,\boldsymbol{u}_t} \end{bmatrix} = H_g(\hat{\boldsymbol{x}}_t, \hat{\boldsymbol{u}}_t) + J_f^\top(\hat{\boldsymbol{x}}_t, \hat{\boldsymbol{u}}_t) \boldsymbol{R}_{t+1} J_f(\hat{\boldsymbol{x}}_t, \hat{\boldsymbol{u}}_t) \\ \begin{bmatrix} \bar{\boldsymbol{P}}_{\boldsymbol{x}_t} \\ \bar{\boldsymbol{P}}_{\boldsymbol{u}_t} \end{bmatrix} = J_g(\hat{\boldsymbol{x}}_t, \hat{\boldsymbol{u}}_t) + J_f^\top(\hat{\boldsymbol{x}}_t, \hat{\boldsymbol{u}}_t) \boldsymbol{r}_{t+1} \\ \boldsymbol{R}_t = \bar{\boldsymbol{Q}}_{\boldsymbol{x}_t,\boldsymbol{x}_t} + \bar{\boldsymbol{Q}}_{\boldsymbol{x}_t,\boldsymbol{u}_t} \boldsymbol{K}_t + \boldsymbol{K}_t^\top \bar{\boldsymbol{Q}}_{\boldsymbol{u}_t,\boldsymbol{x}_t} + \boldsymbol{K}_t^\top \bar{\boldsymbol{Q}}_{\boldsymbol{u}_t,\boldsymbol{u}_t} \boldsymbol{K}_t \\ \boldsymbol{r}_t = \bar{\boldsymbol{P}}_{\boldsymbol{u}_t} + \bar{\boldsymbol{Q}}_{\boldsymbol{x}_t,\boldsymbol{u}_t} \boldsymbol{v}_t + \boldsymbol{K}_t^\top \bar{\boldsymbol{Q}}_{\boldsymbol{u}_t,\boldsymbol{u}_t} + \boldsymbol{K}_t^\top \bar{\boldsymbol{Q}}_{\boldsymbol{u}_t,\boldsymbol{u}_t} \boldsymbol{v}_t \end{cases}, \tag{61}$$

and terminal conditions are $\boldsymbol{K}_n = -\boldsymbol{Q}_{\hat{\boldsymbol{u}}_n,\hat{\boldsymbol{u}}_n}^{-1} \boldsymbol{Q}_{\hat{\boldsymbol{u}}_n,\hat{\boldsymbol{x}}_n}$ *and* $\boldsymbol{v}_n = \boldsymbol{Q}_{\hat{\boldsymbol{u}}_n,\hat{\boldsymbol{u}}_n}^{-1} \boldsymbol{P}_{\boldsymbol{u}_n}$.

A.2 Application in the problem

Based on the implement of iLQR(Appendix A.1), we define:

$$\dot{\bar{\boldsymbol{\Lambda}}}(t) = (\boldsymbol{A}_y \boldsymbol{G}(0) - \beta_k I) \bar{\boldsymbol{\Lambda}}(t) = f(\bar{\boldsymbol{\Lambda}}(t), \boldsymbol{y}_t) \tag{62}$$

$$\min_{\boldsymbol{y}_{1:n}} \sum_{i=1}^{n} C_i(\bar{\boldsymbol{\Lambda}}(t + i\Delta t), \boldsymbol{y}_i) = \min_{\boldsymbol{y}_{1:n}} \sum_{i=1}^{n} g(\boldsymbol{x}_t, \boldsymbol{u}_t) \tag{63}$$

Firstly, we need to give a reference trajectory $(\hat{\boldsymbol{x}}_t, \hat{\boldsymbol{u}}_t)$, which is $(\bar{\boldsymbol{\Lambda}}(t), \boldsymbol{y}(t))$. It may be convenience to set $\boldsymbol{y}_t = \boldsymbol{0}$ and run a trajectory of $\bar{\boldsymbol{\Lambda}}(t)$ as the reference.
Then, J_f is the Jacobi matrix of system dynamic,

$$J_f(\hat{\boldsymbol{x}}_t, \hat{\boldsymbol{u}}_t) = \begin{bmatrix} \boldsymbol{A}_t & \boldsymbol{B}_t \end{bmatrix} = \begin{bmatrix} \frac{\partial f(\bar{\boldsymbol{\Lambda}}(t), \boldsymbol{y}_t)}{\partial \bar{\boldsymbol{\Lambda}}(t)} & \frac{\partial f(\bar{\boldsymbol{\Lambda}}(t), \boldsymbol{y}_t)}{\partial \boldsymbol{y}(t)} \end{bmatrix} \tag{64}$$

where noted that we need to save values of it on different reference points, such as $(\bar{\boldsymbol{\Lambda}}(1), \boldsymbol{y}(1))$, $(\bar{\boldsymbol{\Lambda}}(2), \boldsymbol{y}(2))$, $\ldots$, $(\bar{\boldsymbol{\Lambda}}(n), \boldsymbol{y}(n))$. Similarly, we need to derive and save Jacobi and Hessian matrices of cost function g on reference point.

$$J_g(\hat{\boldsymbol{x}}_t, \hat{\boldsymbol{u}}_t) = \begin{bmatrix} \boldsymbol{P}_{\boldsymbol{x}_t} & \boldsymbol{P}_{\boldsymbol{u}_t} \end{bmatrix}^\top = \begin{bmatrix} \frac{\partial g(\bar{\boldsymbol{\Lambda}}(t), \boldsymbol{y}_t)}{\partial \bar{\boldsymbol{\Lambda}}(t)} & \frac{\partial g(\bar{\boldsymbol{\Lambda}}(t), \boldsymbol{y}_t)}{\partial \boldsymbol{y}(t)} \end{bmatrix} \tag{65}$$

$$H_g(\hat{\boldsymbol{x}}_t, \hat{\boldsymbol{u}}_t) = \begin{bmatrix} \boldsymbol{Q}_{\boldsymbol{x}_t, \boldsymbol{x}_t} & \boldsymbol{Q}_{\boldsymbol{x}_t, \boldsymbol{u}_t} \\ \boldsymbol{Q}_{\boldsymbol{u}_t, \boldsymbol{x}_t} & \boldsymbol{Q}_{\boldsymbol{u}_t, \boldsymbol{u}_t} \end{bmatrix} = \begin{bmatrix} \frac{\partial^2 g(\bar{\boldsymbol{\Lambda}}(t), \boldsymbol{y}_t)}{\partial \bar{\boldsymbol{\Lambda}}(t)^2} & \frac{\partial g(\bar{\boldsymbol{\Lambda}}(t), \boldsymbol{y}_t)^2}{\partial \bar{\boldsymbol{\Lambda}}(t) \partial \boldsymbol{y}(t)} \\ \frac{\partial g(\bar{\boldsymbol{\Lambda}}(t), \boldsymbol{y}_t)^2}{\partial \partial \boldsymbol{y}(\bar{t}) \boldsymbol{\Lambda}(t)} & \frac{\partial^2 g(\bar{\boldsymbol{\Lambda}}(t), \boldsymbol{y}_t)}{\partial \boldsymbol{y}(t)^2} \end{bmatrix}. \tag{66}$$

In practical, parameters of LQR and iLQR is solved by backward. It means that the iteration of $\bar{\boldsymbol{Q}}$, $\bar{\boldsymbol{P}}$, and other parametes are solved from the terminal to the current, because we only have terminal conditions.

A.3 Recursive form of constant input

$$\boldsymbol{\Lambda}(t+\delta) = \boldsymbol{\mu} + e^{-\beta\delta}[\boldsymbol{\Lambda}(t) - \boldsymbol{\mu}] + \boldsymbol{A}(\boldsymbol{y}) \int_t^{t+\delta} g(t+\delta-\tau)\boldsymbol{\Lambda}(\tau) d\tau \tag{67}$$

Similarly, taking $\delta \Rightarrow 0$ and it yields that

$$\dot{\boldsymbol{\Lambda}}(t) = [\boldsymbol{A}(\boldsymbol{y}) - \beta I]\, \boldsymbol{\Lambda}(t) + \beta\boldsymbol{\mu} \tag{68}$$

Solving the equation and that

$$\boldsymbol{\Lambda}(t) = e^{[\boldsymbol{A}(\boldsymbol{y}) - \beta \boldsymbol{I}]t} + \beta\, [\boldsymbol{A}(\boldsymbol{y}) - \beta \boldsymbol{I}]^{-1} \left\{ e^{[\boldsymbol{A}(\boldsymbol{y}) - \beta \boldsymbol{I}]t} - \boldsymbol{I} \right\} \boldsymbol{\mu} \tag{69}$$

References

R. T. Chen, B. Amos, and M. Nickel. Neural spatio-temporal point processes. *arXiv preprint arXiv:2011.04583*, 2020.

J. Jia and A. R. Benson. Neural jump stochastic differential equations. *Advances in Neural Information Processing Systems*, 32, 2019.

Z. J. Li. Reinforcement learning for traffic signal control. *Traffic Signal Control Open Datasets*, 2018. URL https://traffic-signal-control.github.io/.

NY-Times. Coronavirus (covid-19) data in the united states. 2020. URL https://github.com/nytimes/covid-19-data.