HIGH PERFORMANCE NEURAL JUMP STOCHASTIC DIFFERENTIAL EQUATIONS

SHIJIE ZHANG*

Abstract. Many real-world systems evolve continuously over time, but there are also stochastic jumps that will interrupt the continuous flow. Describing how these stochastic jumps affect the continuous dynamics is often a challenging task. Jia et al. (2019)[2] extend the Neural Ordinary Differential Equations (Neural ODEs) introduced by Chen et al. (2018)[1] by incorporating a discrete stochastic jump term. The Neural Jump Stochastic Differential Equations (Neural JSDEs) they created could help us to learn hybrid systems that contains both continuous and discrete behavior. In this project, I review the algorithm of Neural JSDEs and develop a high performance implementation on Julia. Then I apply this framework to model a process that combines continuous process with an exponential Hawkes process. The Neural JSDEs can help researchers to understand the real-world systems that combines continuous and discrete dynamics, thereby facilitating advancements in various fields of study, such as finance, epidemiology, seismology and biology.

Key words. High Performance Computing, Neural Network, Stochastic Differential Equation, Stochastic Jumps.

AMS subject classifications. 65C30, 60H15

1. Introduction.

A significant portion of real-world problems evolves continuously over time. We normally describe such continuous systems by using Ordinary Differential Equations(ODEs) or Partial Differential Equations(PDEs) and researchers have extensively worked on identifying the governing equations for specific continuously evolving systems, as well as analyzing and solving ODEs and PDEs to understand the dynamics of the system.

However, some problems is not always continuously evolving and may be interrupted by stochastic events. These hybrid systems are not uncommon. For example, consider a game of tennis: the ball follows a continuous trajectory but will abruptly change its moving trajectory when struck by a racket. Similar hybrid evolving systems can also be found in various scientific domains. A cell undergoes continuous growth, with its size function exhibiting continuity. However, when the cell reaches a certain size or at a specific time point, the cell will divide into two daughter cells and the size of each daughter is only half of the mother cell. If we are analyzing the size function, we will see a sudden change. The stochastic jump behavior is important for us to understand these hybrid systems. Unfortunately, we know too little about how to analyze the stochastic jumps.

Here, I review the Neural Jump Stochastic Differential Equations (Neural JSDEs) proposed by Jia et al. (2019)[2] to to effectively capture and understand the dynamics of continuous and discrete hybrid systems. To represent the state of the hybrid system, I employ a latent vector $\mathbf{z}(t) \in \mathbb{R}^n$, which will evolves continuously until a stochastic event occurs. Upon the occurrence of such an event, the trajectory of the latent vector $\mathbf{z}(t)$ will have a abrupt jump, after which it resumes continuous evolution until the next stochastic event takes place. The continuous part of $\mathbf{z}(t)$ follows the model of Neural Ordinary Differential Equations (Neural ODEs) introduced by Chen et al. (2018)[1]. This approach utilizes a neural network to parameterize the continuous flow dynamics. While the stochastic jump behavior is describede by another neural network. These two neural networks would be implemented on Julia using GPU

*Department of Mechanical Engineering, Cambridge, MA (shjzhang@mit.edu).

acceleration in this project.

The Neural ODEs framework draws inspiration from residual networks and we could calculate the derivative of its loss function using adjoint method. This approach is highly effective to describe systems that exhibit continuous evolution. However, it lacks the capability to depict discrete events that cause abrupt changes in the continuous trajectory. To address this limitation, the Neural JSDEs extends the continuous framework of Neural ODEs by incorporating stochastic jumps, which both continuous dynamics and discrete jumps. The ability of the Neural JSDEs to model continuous and discrete hybrid dynamics makes it become a powerful tool for modeling systems with hybrid dynamics that combine continuous and discrete behaviors.

2. Point Processs. Point processes are discrete stochastic models used to describe the random occurrence of points or events. The time sequence when the discrete events happen could be described as a set of time $\mathcal{H} = \{\tau_j\}$ and the function $N(t)$ could be used to describe the total number of events occurred before time t:

$$N(t) = \sum_{\tau_j \in \mathcal{H}} H(t - \tau_j) \tag{2.1}$$

where H is the Heaviside step function.

Among point processes, what we are mostly interested in is the temporal point processes, that is, the occurrence of future events depend on past events. Temporal point processes have wide applications in various fields, including finance, epidemiology, telecommunications, and social sciences. The dependency of future events on past events could be described as a conditional probability on $\mathcal{H} = \{\tau_j\}$, which is the set of hitorical events. We use a function $\lambda(t)$ to describe the conditional intensity at time t. Then, we could describe the probability that event happen in $[t, t+dt)$ as:

$$\mathbf{P}\{\text{event happen in } [t, t+dt)|\mathcal{H}_t\} = \lambda(t) \cdot dt \tag{2.2}$$

2.1. Hawkes Process. One of the most well-studied and commonly used point processes is the Hawkes processes, which is also what I later used in this project. The Hawkes process is a kind of self-exciting process, which means a past event will leave an impact on the probability of future events. That is conditional intensity function $\lambda(t)$ after it happens. The intensity function would be:

$$\lambda(t) = \lambda_0 + \alpha \sum_{\tau_j \in \mathcal{H}_t} \kappa(t - \tau_j) \tag{2.3}$$

here, λ_0 is the baseline intensity, κ is a kernel function. There are two widely used kernels for Hawkes Processthe exponential kernel κ_1 and the power-law kernel κ_2:

$$\kappa_1(t) = e^{-\beta t} \tag{2.4}$$

$$\kappa_2(t) = \begin{cases} 0, & t < \sigma \\ \frac{\beta}{\sigma}(\frac{t}{\sigma})^{-\beta-1}, & otherwise \end{cases}$$

In this project, I consider the exponential Hawkes process and apply Neural JSDEs on this process.

3. Neural Jump Stochastic Differential Equation. In the Neural JSDEs, we represents the latent state of the continuous and discrete hybrid system with a vector $\mathbf{z}(t) \in \mathbb{R}^n$, that is , the latent state evolves continuously with a deterministic trajectories and will be interrupted by discrete stochastic jumps. Then, the latent state dynamics could be described as following:

$$d\mathbf{z}(t) = f(\mathbf{z}(t), t; \theta) \cdot dt + w(\mathbf{z}(t), t; \theta) \cdot dN(t) \tag{3.1}$$

here, $f(\mathbf{z}(t), t; \theta)$ and $w(\mathbf{z}(t), t; \theta)$ are neural networks that control the flow and jump. $N(t)$ is the total number of the occurrence of events up to time t (see Equation 2.1).

Then, we could simulate the dynamics of the continuous and discrete hybrid system by integrating Equation 3.1. Details of simulating the hybrid system can be seen in Section 5.1.

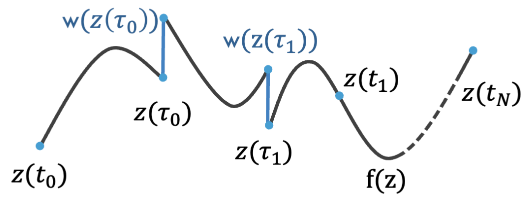

FIG. 1. *Forward-mode of the jump stochastic differential equation 3.1. Here, $f(z)$ and $w(z(\tau_i))$ are neural networks that control the flow and jump respectively.*

Then the transformed state at any time $\tau_j < t_i < \tau_{j+1}$ is given by integrating the ODE forward from time τ_j:

$$\mathbf{z}(t_i) = \mathbf{z}(\tau_j^+) + \int_{\tau_j}^{t_i} f(\mathbf{z}(t), t; \theta) dt \tag{3.2}$$

Noting that the right limit of the latent vector $\mathbf{z}(t)$ is $\mathbf{z}(t^+) = lim_{\epsilon \to 0} \mathbf{z}(t + \epsilon)$. Then, at each timestep τ_j when event happens, the latent state would be:

$$\mathbf{z}(\tau_j^+) = \mathbf{z}(\tau_j) + w(\mathbf{z}(\tau_j), \tau_j; \theta) \tag{3.3}$$

Then, we consider optimizing a scalar-valued loss function L, whose input is the result of forward propagation. At continuous timestep $\tau_j < t_i < \tau_{j+1}$, the derivative of the loss function can be calculated by the adjoint method used in Chen et al. (2018)[1]:

$$\frac{d\mathbf{a}(t)}{dt} = -\mathbf{a}(t) \frac{\partial f(\mathbf{z}(t), t; \theta)}{\partial \mathbf{z}(t)} \tag{3.4}$$

$$\frac{d\mathbf{a}_\theta(t)}{dt} = -\mathbf{a}(t) \frac{\partial f(\mathbf{z}(t), t; \theta)}{\partial \theta}$$

$$\frac{d\mathbf{a}_t(t)}{dt} = -\mathbf{a}(t) \frac{\partial f(\mathbf{z}(t), t; \theta)}{\partial t}$$

Integrating Equation 3.4 backward in time, we could get:

$$\mathbf{a}(t_i) = \mathbf{a}(\tau_{j+1}) + \int_{\tau_{j+1}}^{t_i} \left[\frac{d\mathbf{a}(t)}{dt} - \sum \delta(t - t_i) \frac{\partial L}{\partial \mathbf{z}(t_i)} \right] dt \tag{3.5}$$

$$\mathbf{a}_\theta(t_i) = \mathbf{a}_\theta(\tau_{j+1}) + \int_{\tau_{j+1}}^{t_i} \frac{d\mathbf{a}_\theta(t)}{dt} dt$$

$$\mathbf{a}_t(t_i) = \mathbf{a}_t(\tau_{j+1}) + \int_{\tau_{j+1}}^{t_i} \left[\frac{d\mathbf{a}_t(t)}{dt} - \sum \delta(t - t_i) \frac{\partial L}{\partial t_i} \right] dt$$

At discontinuity point τ_j, the adjoint sensitivity variable would satisfy:

$$\mathbf{a}(\tau_j) = \mathbf{a}(\tau_j^+) \frac{\partial \mathbf{z}(\tau_j^+)}{\partial \mathbf{z}(\tau_j)} \tag{3.6}$$

So, we will have:

$$\mathbf{a}(\tau_j) = \mathbf{a}(\tau_j^+) + \mathbf{a}(\tau_j^+) \frac{\partial [w(\mathbf{z}(\tau_j), \tau_j; \theta)]}{\partial \mathbf{z}(\tau_j)} \tag{3.7}$$

$$\mathbf{a}_\theta(\tau_j) = \mathbf{a}_\theta(\tau_j^+) + \mathbf{a}(\tau_j^+) \frac{\partial [w(\mathbf{z}(\tau_j), \tau_j; \theta)]}{\partial \theta}$$

$$\mathbf{a}_t(\tau_j) = \mathbf{a}_t(\tau_j^+) + \mathbf{a}(\tau_j^+) \frac{\partial [w(\mathbf{z}(\tau_j), \tau_j; \theta)]}{\partial \tau_j}$$

Then, we could compute the derivative of the loss function $\frac{dL}{d\mathbf{z}(t_0)} = \mathbf{a}(t_0)$, $\frac{dL}{d\theta} = \mathbf{a}_\theta(t_0)$, $\frac{dL}{dt_0} = \mathbf{a}_t(t_0)$ by calculating backward starting with the final value of $\mathbf{z}(t_N)$ by using Equation 3.5 abd Equation 3.7. The initial condition of the adjoint variables would be :

$$\mathbf{a}(t_N) = \frac{\partial L}{\partial \mathbf{z}(t_N)} \tag{3.8}$$

$$\mathbf{a}_\theta(t_N) = 0$$

$$\mathbf{a}_t(t_N) = \frac{\partial L}{\partial t_N} = \mathbf{a}(t_N) f(\mathbf{z}(t_N), t_N; \theta)$$

Details of the backward propagation could be find in Section 5.2.

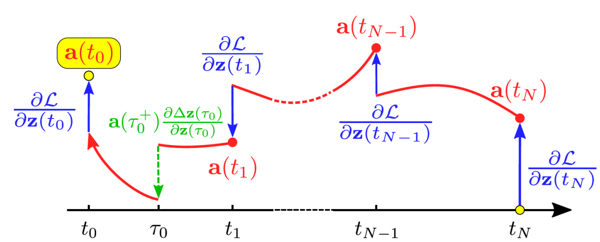

FIG. 2. *Backward-mode of the jump stochastic differential equation 3.1. Algorithm to Calculate the loss function derivative from the final value of the latent vector $\mathbf{z}(t_N)$ follows Equation 3.5 and Equation 3.7.*

4. Experiments and Results.

In this project, I perform the Neural JSDE on a continuous flow affected by an exponential Hawkes Process. For simplicity, I implement it on a 1-Dimensional system, i.e. $\mathbf{z}(t) \in \mathbb{R}, t \in (0, 100)$. I set the true value of the continuous trajectory as:

$$f(\mathbf{z}(t), t; \theta) = 1 \tag{4.1}$$

The true value of the conditional intensity of the discrete event would follows the exponential Hawkes Process:

$$\lambda(t) = \lambda_0 + \alpha \sum_{\tau_j \in \mathcal{H}_t} e^{-\beta(t-\tau_j)} \tag{4.2}$$

In this project, I set $\beta = 1, \alpha = 0.8, \lambda_0 = 0.2$. Each time an event happens, $\mathbf{z} = \mathbf{z} + 1$. Then, we could simulate this process using DifferentialEquations.jl in Julia. Then we could get the event sequence $\mathcal{H}$ and the true evolving trajectories. Using this as our ground truth, we implement the Neural JSDEs to this process and get the result, which is quite promising:

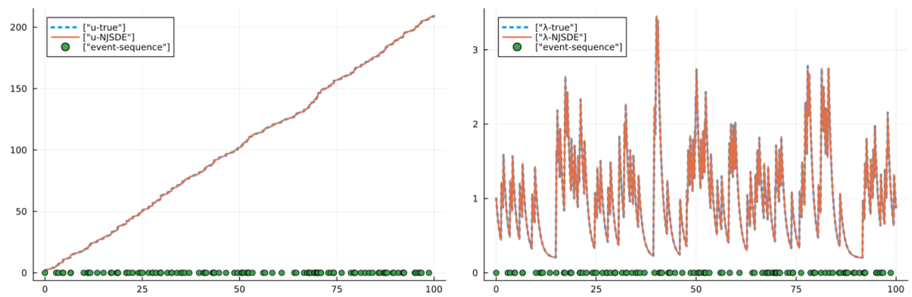

FIG. 3. *Learning results of Neural JSDEs applying on a simple continuously evolving system that interrupted by an exponential Hawkes Process.*

5. Algorithm of Neural JSDEs.

5.1. Forward Propagation. To do forward propagation, we begin with the model parameter θ, which consists of the parameter of the two neural network $f(\mathbf{z}(t), t; \theta)$ and $w(\mathbf{z}(t), t; \theta)$. We also need to the start time t_0, end time t_N, and initial state of the latent vector $\mathbf{z}(t_0)$ as our input.

In this project, I set the two neural network have the same simple structure $NN : \mathbb{R} \to \mathbb{R}$:

$$NN(z; W_i, b_i) = W_2 tanh.(W_1 z + b_1) + b_2 \tag{5.1}$$

where W_1 is 50×1, b_1 is length 50, W_2 is 1×50, b_1 is length 1.

In forward-mode, we need to simulate the timestep that stochastic events happen. So the algorithm would go like this:

Algorithm 5.1 Forward-mode of Neural JSDEs

Initialization: $t = t_0, j = 0, \mathcal{H} = \{\}, \mathbf{z} = \mathbf{z}(t_0)$
while $t \le t_N$ **do**
  τ_j = NextEvent($\mathbf{z}, t, \theta$)
  $\mathbf{z}$ =StepForward($\mathbf{z}, t, \tau_j, \theta$)
  $\mathbf{z}$ =JumpForward($\mathbf{z}, \tau_j, \theta$)
  $\mathcal{H} = \mathcal{H} \cup \{\tau_j\}$
  $j = j + 1$
  $t = t + \tau_j$
end while

In this forward-mode algorithm, StepForward$(\mathbf{z}, t, \tau_j, \theta)$ is an ODE solver that integrate the neural network $f(\mathbf{z}(t), t; \theta)$ from time t to time τ_j, which is the second term in Equation 3.2 (i.e. the first term in Equation 3.1). JumpForward$(\mathbf{z}, \tau_j, \theta)$ is a jump process, which add the value of $w(\mathbf{z}(\tau_j), \tau_j; \theta)$ to $\mathbf{z}(\tau_j)$ to get $\mathbf{z}(\tau_j^+)$. It implements the first term in Equation 3.2 (i.e. the second term in Equation 3.1).

5.2. Backward Propagation. To do backward Propagation to get the derivatives of loss function L: $\frac{dL}{d\mathbf{z}(t_0)} = \mathbf{a}(t_0)$, $\frac{dL}{d\theta} = \mathbf{a}_\theta(t_0)$, $\frac{dL}{dt_0} = \mathbf{a}_t(t_0)$, we begin with the model parameter θ. We also need to the start time t_0, end time t_N, and initial state of the latent vector $\mathbf{z}(t_0)$ and the event sequence $\mathcal{H}$ we get from forward-mode algorithm in Section 5.1 as our input.

Algorithm 5.2 Backward-mode of Neural JSDEs

Initialization: $t = t_0, \mathbf{z} = \mathbf{z}(t_0)$
while $t \leq t_N$ **do**
  $\tau_j =$ NextEvent$(\mathbf{z}, t, \theta)$
  $\mathbf{z} =$StepForward$(\mathbf{z}, t, \tau_j, \theta)$
  $\mathbf{z} =$JumpForward$(\mathbf{z}, \tau_j, \theta)$
  $t = t + \tau_j$
end while
Compute loss function: $L = L(\{\mathbf{z}(t_i)\}, \{\mathbf{z}(\tau_j)\}; \theta)$
while $t \geq t_0$ **do**
  $\tau_j =$ PreviousEvent$(\mathcal{H}, t)$
  $\mathbf{z}, \mathbf{a}, \mathbf{a}_\theta, \mathbf{a}_t =$StepBackward$(\mathbf{z}, \mathbf{a}, \mathbf{a}_\theta, \mathbf{a}_t, t, \tau_j, \theta)$
  $\mathbf{z} =$JumpBackward$(\mathbf{z}, \mathbf{a}, \mathbf{a}_\theta, \mathbf{a}_t, \tau_j, \theta)$
  $t = \tau_j$
end while

In this backward-mode algorithm, StepBackward$(\mathbf{z}, \mathbf{a}, \mathbf{a}_\theta, \mathbf{a}_t, t, \tau_j, \theta)$ is an ODE solver that integrate the backward ODEs (Equation 3.4), which is Equation 3.5. JumpBackward$(\mathbf{z}, \mathbf{a}, \mathbf{a}_\theta, \mathbf{a}_t, \tau_j, \theta)$ is a jump process, which implements Equation 3.7.

5.3. High Performance settings. In this project, I accelerate the Neural JSDEs using GPU. I use CuArrays.jl to make underlying array type be able to be accelerated by GPU. By using CuArrays.jl, the neural network in Neural JSDEs algorithm is accelerated.

6. Discussion. In this project, I have implemented a high-performance version of Neural JSDEs on GPU using CuArrays.jl package in Julia, which could enhance the computational efficiency of the Neural JSDEs algotrithm. To test the algorithm, I apply the Neural JSDEs framework to an exponential Hawkes process, which is well-studied and high applicable in real scenarios, and the results are quite promising. While this project focuses on a temporal point process, the high-performance Neural JSDEs algorithm has broad applicability across various fields. The algorithm is not limited to well-studied processes like the exponential Hawkes process or other known stochastic point processes. It can also be extended to model other unknown jump stochastic differential equations, providing a data-driven approach to understanding such commonly seen systems in real-world. The Neural JSDEs algorithm allows us to simulate and predict the discrete events that stochastically occurs in a continuous dynamics.

In real-world problems, we often encounter problems that combine a continuous flow and stochastic jumps. The Neural JSDEs could help us understand and analyze such complex dynamics. By applying it to real experimental data, we can simulate and predict unknown processes. The versatility of Neural JSDEs holds significant potential for advancing our understanding of continuous and discrete hybrid systems and their behaviors.

7. Code Accessibility. The complete implementation of our algorithms and exponential Hawkes process experiments is available at https://github.com/ShijieZhang10/18$_3$37$_N JSDE$. Codes and figures used in this project can also be found in this repository.

REFERENCES

[1] R. T. Chen, Y. Rubanova, J. Bettencourt, and D. K. Duvenaud, *Neural ordinary differential equations*, Advances in neural information processing systems, 31 (2018).

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