Proceedings of the 2023 International Conference on Machine Learning and Automation
DOI: 10.54254/2755-2721/37/20230474

Review of common spiking neural network simulation tools

Linyang Li

University of Leeds, Leeds, the United Kingdom, LS2 9JT

lilinyang2020@yeah.net

Abstract. As a recognized third-generation neural network, the spiking neural network has high concurrency and complexity. Although the degree of research is still far from the previous generation of neural networks, spiking neural networks are excellent in performance and energy consumption. In this paper, the common spiking neural network simulation tools are reviewed. The most frequently used and mentioned tools are NEURON, NEST, and BRAIN. NEURON is more suitable for simulation based on biological applications, pays more attention to biological characteristics, and can support large-scale network simulation. Examples used in the official documentation are neural simulations of invertebrates and mammals. Large heterogeneous networks of point neurons or neurons with a few compartments are frequently simulated using NEST. In contrast to models that concentrate on the specific morphological and biophysical characteristics of individual neurons, NEST is appropriate for those that emphasize the dynamics, size, and structure of the nervous system. Brian was originally designed for research and teaching and is well suited as a teaching and presentation tool for simulating and observing the effects of different parameters for classical neural network projects such as picture classification. In addition, CSIM, SPLIT, SPINNAKER, and other tools also have their merits, but due to the low frequency of relevant references and lack of universality, this study will not give a detailed introduction.

Keywords: Spiking Neural Network, SNNsimulation, NEURON, NEST, BRAIN.

1. Introduction

Spiking neural networks have the advantages of strong anti-interference and anti-noise abilities. It is now shining in many fields besides robotics and artificial intelligence. For example, speech recognition, image recognition, face recognition, target recognition, and tracking in the field of information recognition There is also biological analysis, human brain and cortical modeling, and financial analysis combined with data analysis [1].

In fact, the complexity of the real biological neural network system is extremely high and cannot be fully simulated by the current technical means. Human neural networks and even subsystems contain millions of neurons, each of which has ion channels, impulse responses, and other properties.

There are many things to consider when simulating neural networks. For example, there are clock-driven simulation strategies, event-driven simulation strategies, voltage-driven simulation strategies, and so on. This makes it more difficult for beginners who only want to experience it.

At present, there are many choices of neural network simulation software, which greatly reduces the difficulty of simulation, and it is no longer necessary to build a simulation environment yourself. However, different simulation tools have different original intentions and different learning costs. For

DOI: 10.54254/2755-2721/37/20230474

example, mastering simulation tools requires learning the method parameters of different tools and their chosen scripting languages, such as Neuron's Hoc and NMODL, NEST's SLI, and Genesis's SLI. Therefore, this paper summarizes the characteristics of different tools. This paper is conducive to more scholars understanding and using neural networks and provides a reference for beginners.

2. Introduction to simulation tools

When simulating neurons, the general steps are as follows: First, obtain the internal state variables and updated values of each neuron. Second, detect whether the neuron emits pulses and record the spike time. Third, propagating pulses in neural networks 4 processing the propagated pulses [2].

Follow the general steps. This section will introduce the following tools:

2.1. NEURON

NEURON is the result of research at Yale University in 1997. It is designed to display the electrical and chemical signals in real neurons and neural networks, aiming to create more convenient models that simulate biological brain mechanisms and use these models to solve real problems. The official documentation uses invertebrates and mammals as reference paradigms and points out that NEURON focuses on the interface geometry of cells when performing simulations.

First, NEURON is flexible and fast, mainly because its model allows users to decide which details to keep and which to ignore. Secondly, because the membrane voltage is calculated by an implicit integration method optimized for the branch structure, NEURON is very fast. Moreover, as the complexity of the morphology and membrane mechanism rises, the performance can keep decreasing very slowly. Additionally, it has been utilized with very huge network models.

Another great feature of NEURON is the implementation of functions that implement a fully graphical, windowing interface. Users are able to easily develop and configure menu displays, editors for parameter values, graphs of parameters and state variables, and views of model neurons through this interface without writing any code. Saving users' learning costs, the user-defined mechanisms in NMODL are automatically converted to C, compiled and linked to other languages. It also uses a built-in object-oriented interpreter for defining neuron morphology and membrane properties, controlling simulations, and establishing the appearance of graphical interfaces. It defaults to a graphical interface for exploratory simulations, including setting parameters, voltage and current stimulation controls, and graphical variables such as cell geometry and cross-sectional shape as a function of time and position.

It appeared in 1997 but is still maintained and widely used. The last update to date, version 8.2, was released on July 1, 2022. It is mainly developed for biological research, and the neuron is a very suitable choice when there is a need for biological research [2].

2.2. NEST

NEST was created by the Neural Simulation Technology Initiative, a company established in 2001 to advance and document simulation techniques for large neural networks. NEST is best suited for models that concentrate on the dynamics, size, and structure of neural systems rather than the specific morphological and biophysical characteristics of individual neurons. Current case studies in which the simulation tool has been used include: models of sensory processing, such as the mammalian visual or auditory cortex; models of network activity dynamics, such as laminar cortical networks; spike synchronization models in feed forward networks, such as Synfire Chains, etc.

In a NEST, a network is a group of nodes and the connections between them. Neurons, devices, etc. can be used as nodes. The types of events that nodes send and receive can also be different. NEST provides a number of models for users to use. These include Integrate-and-fire neuron models, Hodgkin-Huxley models, and Models for static synapses. NEST has the ability to build large networks. The existence of subnetworks helps large network models organize themselves into manageable parts [3].

DOI: 10.54254/2755-2721/37/20230474

The accuracy of NEST is also worth mentioning. There is no global equation solver used by NEST. In a heterogeneous network, each node may have its own set of state equations, allowing each node to update its state using the most precise equation technique. Meanwhile, for a large class of neuron models, NEST integrates the dynamic equations with machine precision, using a technique termed Exact Integration in the context of pulse coupled neuronal systems [4]. This method integrates sets of linear time-invariant differential equations using the matrix exponential.

The key factor contributing to NEST's speed advantage is its capacity to expand memory availability or accelerate simulations by utilizing multiprocessor processors and computer clusters. NEST scaled more effectively than linearly in parallel simulations [5].

NEST is very malleable and has a modular architecture. For the particular issues they want, users can create their own models for NEST. NEST can also be extended in a variety of ways: at the simulation kernel level, new models can be added to neurons and devices. At the programming language level, functions and libraries can be added. Examples are PyNEST and PyNN in Python. NEST's native simulation language interpreter simulation language module is SLI. In addition, NEST simulation results can also be written to files for subsequent analysis using tools such as Matlab and Mathematica.

The NEST Initiative releases regular software updates, which are available directly at http://www.nest-initiative.org. Linux, Apple, Sun Solaris, HP Tru64, Windows, and other operating systems have all seen success with NEST. The official advice is to first install Linux on the virtual machine before adding NEST.

The NEST simulator project was last updated on January 30, 2013. Tutorial page (NEural_Simulation_Tool:http://www.scholarpedia.org/article/NEST) has been visited 124,815 times, it is also a stable and widely used simulation tool.

2.3. BRAIN

Brian is creating a clock-driven spiking neural network simulator. It has a tighter integration with Python is a user-friendly and incredibly adaptable tool, especially for single-compartment neural networks, for creating new models quickly. Originally designed for research and teaching, BRAIN is well suited to be a teaching and presentation tool. The advantages of BRAIN are very significant, mainly as follows:

Easy to get started.Users of Brian just need to become proficient in one programming language. The way BRAIN functions is by offering Python packages comprising objects, classes, and functions. It can be used interactively via the Python shell or as a component of a Python program. First off, learning the Python language is quicker and simpler to demonstrate examples thanks to the syntax's ability to produce relatively little amounts of code. Second, there are fewer compatibility difficulties for students with diverse hardware or operating systems because BRAIN was developed in pure Python and is accessible on practically all platforms. Finally, using Brian is relatively simple, and the syntax and basic concepts of the Brian programming language are remarkably akin to those of neuroscience.

Brian's complementary projects brought some unique features to BRAIN, making it more flexible. First, equations, especially differential equations, can be defined at the highest level using standard mathematical notation. Second, Brian doesn't restrict you to using standard models of neurons and synapses (although the library provides many).

Also because the code is implemented in Python rather than in a separate language for the interface. Because of this, Brian's use is more versatile., such as by writing code to read and modify simulation variables while the simulation is running. It works very efficiently.

Despite Python as an interpreted language and each Python operation incurs overhead, BRAIN is not inefficient and can still be simulated efficiently using vectorization techniques. Brian used vectorization for model simulation and building (e.g., initialization of synaptic weights). The time spent on mathematical operations takes substantially longer for big networks than the time spent on interpretative operations. such that the interpretation overhead is negligible. It can nonetheless operate at a speed that is comparable to C code written natively and generally outperforms Matlab code. The interpretation overhead is proportionally much larger for smaller networks, but in many cases it is not important

DOI: 10.54254/2755-2721/37/20230474

because the total time of the simulation is also shorter. So the most unfavorable scenario for Brian is to simulate a small network over a very long biological time. Developers have begun to use the parallel processors found in modern graphics cards to speed up Brian simulations without additional work on the part of the user. These can be used as parallel coprocessors for vectorized computations that can already model spike time dependence, plasticity, and short-term plasticity.

Brian is in his teens, so you can rely on him to give accurate results. And BRAIN has an extensive test suite that can catch any errors such as code errors or instabilities of the solver specified by the difference equations, ensuring that it works on all platforms. Currently developers release a new version about every six months.

The Brian project is open source. Contributors are encouraged by the developers, who choose to use as many existing packages and components as possible and write only what is necessary on top of them. The complete model of any program written using BRAIN can be published to ModelDB. In addition, there is a brand-new "computational neuroscience" project on the NeuralEnsemble website (http://neuralensemble.org/cookbook). These items can be directly cut and pasted into code snippets from other codes. It is very convenient to find reference cases and exchange them [6].

2.4. Other Tools

In addition to the above several detailed simulation tools, there are many simulation tools that, due to various factors, affect the article citation rate and are not suitable for personal simulation use, so this paper does not go into too much detail.

CSIM

The CSIM simulator is written in C++ and is designed to simulate networks containing orders of magnitude up to a few thousand neurons and over 1.000.000 synapses. It develops tools for assessing the computing capacity of these models in the Matlab environment as well as computer models for microcircuits of different sizes and levels of detail. [7].

SPLIT

The SPLIT project, published in 1998 in the Journal of Computational Neuroscience, is a large-scale neural network simulation on multiple hardware platforms. This paper describes the experience gained when applying an existing simulator, SWIM, to two very different architectures: vector computers and multiprocessor workstations [8].

SPINNAKER

The Spiking Neural Network Architecture (SpiNNaker) project seeks to develop massively parallel million-core computers capable of simulating large-scale spiking neural networks in biological real-world situations. The project has delivered systems with up to 2500 processors and demonstrated a real-time event-driven programming model that supports flexible access to machine resources and enables their use through extensive collaboration around the world [9].

3. Conclusion

This paper introduces the Neural, NEST, and BRAIN simulation tools for beginners and mentions the CSIM, SPLIT, and SPINNAKER simulation tools. Important features of different simulation tools are described. It can be concluded that the current common simulation tools have good performance in terms of efficiency and accuracy. The implementation of simulation tools is different, and the simulation objects are also different, which provides researchers with a variety of choices. Appropriate tools can be selected according to their own research strategies and objectives. For example, it is better to choose BRAIN for education and entry and use the least learning cost to contact SNN. To simulate large, heterogeneous neural networks, try tools like NEST. However, this paper has few suggestions on the selection of different simulation strategies and models. In the future, we will focus on the selection and comparison of simulation tools under different simulation strategies.

References

[1] Lin Xh. (2018). Principle and Application of Spiking Neural Network. Science Press. p18-21 p66

DOI: 10.54254/2755-2721/37/20230474

[2] Hines, M. L., & Carnevale, N. T. (1997). The NEURON simulation environment. Neural computation, 9(6), 1179-1209.
[3] Gewaltig, M. O., & Diesmann, M. (2007). Nest (neural simulation tool). Scholarpedia, 2(4), 1430.
[4] Rotter, S., & Diesmann, M. (1999). Exact digital simulation of time-invariant linear systems with applications to neuronal modeling. Biological cybernetics, 81(5-6), 381-402.
[5] Morrison, A., Mehring, C., Geisel, T., Aertsen, A. D., & Diesmann, M. (2005). Advancing the boundaries of high-connectivity network simulation with distributed computing. Neural computation, 17(8), 1776-1801.
[6] Goodman, D. F., & Brette, R. (2008). Brian: a simulator for spiking neural networks in python. Frontiers in neuroinformatics, 2, 350.
[7] Natschläger, T., Markram, H., & Maass, W. (2003). Computer models and analysis tools for neural microcircuits. Neuroscience databases: a practical guide, 123-138.
[8] Hammarlund, P., & Ekeberg, Ö. (1998). Large neural network simulations on multiple hardware platforms. Journal of Computational neuroscience, 5, 443-459.
[9] Furber, S. B., Galluppi, F., Temple, S., & Plana, L. A. (2014). The spinnaker project. Proceedings of the IEEE, 102(5), 652-665.