SpikeTime: Spiking Neuronal Network Simulation in Julia Language

Russell Jarvis Yeshwanth Bethi Pablo de Abreu Urbizagastegui
Ali Mehrabi

***Abstract*—**

I. Introduction

While there is much focus on hardware advances that accellerate the simulation of large scale spiking neural networks, it is worthwhile to shift our attention to language advances that may also support accelerated large scale spiking neural network simulation. Some gains in biologically faithful neuronal network simulation can be achieved by applying recent computer language features. For example, the Julia language supports Sparse Compressed Arrays, Static Arrays, furthermore Julia provides very extensive support for CUDA GPU, as well as a plethora of reduced precision types. Julia also provides a high-level syntax that facilitates high code reuse while simplifying plotting and data analysis. These features lend themselves towards high-performance large-scale Spiking Neural Network simulation. Therefore, we are using Julia to develop an open-source software package that enables the simulation of networks with millions to billions of synapses on a computer with a minimum of 64GB of memory and an NVIDIA GPU.

Another major advantage implementing SNN simulations in the Julia language is reduced technical debt. The simulation code we are developing is both faster and less complicated to read compared with some other simulation frameworks. The simplicity of the code base encompasses a simple installation process. Ease of installation is an important part of neuronal simulators that is often overlooked when evaluating simulation merit, GPU simulation environments are notoriously difficult to install and this big technical burden of installation harms model portability and reproducibility. The Julia language facilitates the ease of installation to solve the "two language problem" of scientific computing. The simulator encompasses a singular language environment, which includes a reliable, versatile, and monolithic package manager. Furthermore the simulator installation includes no external language compilation tools or steps.

To demonstrate the veracity and performance of this new simulation approach, we compare the the Potjans and Diesmann model as implemented in the NEST and GENN simulators. In a pending analysis, we compare simulation execution speeds and spike train raster plots to NEST and GENN using the discussed models as benchmarks. A review of the literature suggests that there is a desire to modernize pre-existing large scale network simulators, but such efforts fall short of re-writing existing simulator code in the Julia language. [1]

The discussed code repository started from using a the pre-existing GitHub code base [2], and is similar in other ways to [3] and [4]

II. Theoretical Framework

We use the forward Euler implementations of synaptic current weight updates, and

$$V_M \tag{1}$$

updates, as the forward Euler method is fast, and sufficiently robust for use on well known homogenuous Leaky Integrate and Fire neurons.

III. Methodological Framework

Forward Euler Weight update rules, and membrane potential update rules where applied.

A. *Virtual Experiment Simulation Protocol*

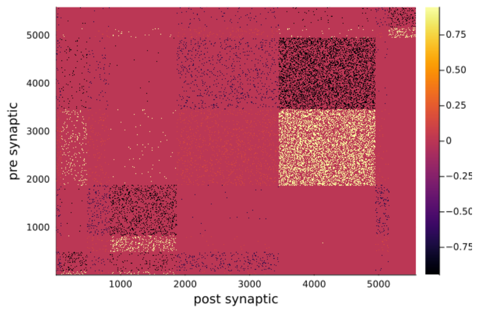
We presented neuromorphic data to the potjan's network model. To do this we constructed a 2D population layer, and gave every cell spatial coordinates. We systematically populated cell centres along a 2D grid.

We then used a distance dependent wiring rule such that neighbouring cells are more likely to be synaptically connected with each other with inhibitory synapses, according to a "winner-take-all" connection scheme. This 2D sheet population of cells then mapped onto regular 1D populations of cells, in the Potjan's balanced E/I model.

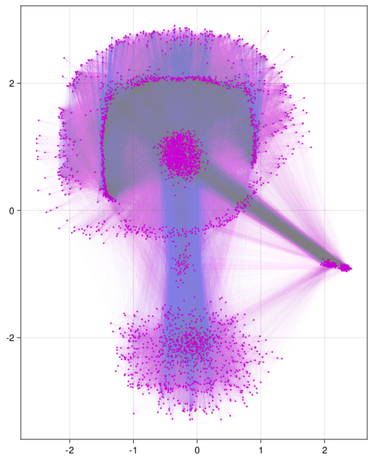
In this way we were able to present several neuromorphic data types to the Potjan's E/I network, and we allowed the network to train its synapses with STDP synaptic update rules. under exposure to 5 out of 10 different numbers of the NMNIST neuromorphic data set. We used the spike2vec algorithm from to show that presenting familiar training items (0-2-4-6-8) to the trained model caused the network to enter attractors (caused the network to recall a repeated temporal spatial pattern). Presenting un-familiar NMNIST items (odd numbers 1-3-5-7-9), did not cause the network to enter those attractors as frequently.

We also applied the Recurrence Analysis metrics to its found state transition matrices, for spike2vec encoded raster plot evolutions under both familiar and unfamiliar item categories. Finally we converted the spike sequence vectors to word vectors, on trained familiar items, and trained unfamiliar items. Converting spikes to words via vectors, allowed us to compare model spike trains with emperical data, using the statistics of natural language [4].

IV. Result Analysis



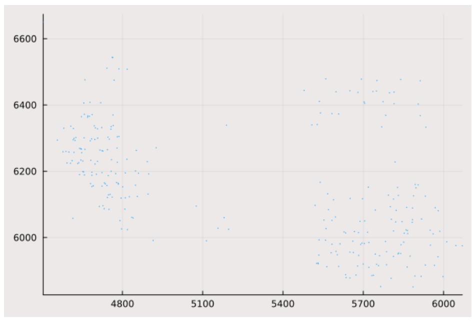
Heatmap visualization of the Potjans and Diesmon static connectome [5], scalled such that the total number of cells including the combined contribution of E and I populations is 5450.



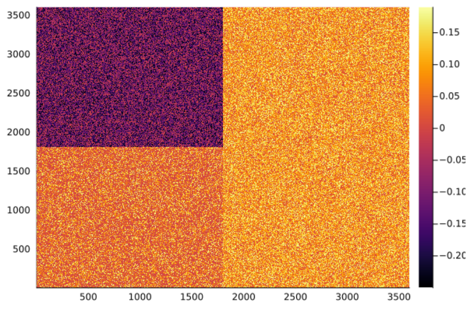
SGtSNEpi visualization of the Potjans and Diesmon static connectome [5]. Graph partitioning of the connectome adjacency matrix can be used to compile network models in a way that minimises spike traffic between GPU thread locks. Although the SGtSNEpi dimensionality reduction technique provides a nice over view of network structure at scale, it is not as fast or useful as other techniques that distribute the network based on effective connectivity measures. different technique we developed which is called Spike2Vec.

V. Validation of Network Simulation Results

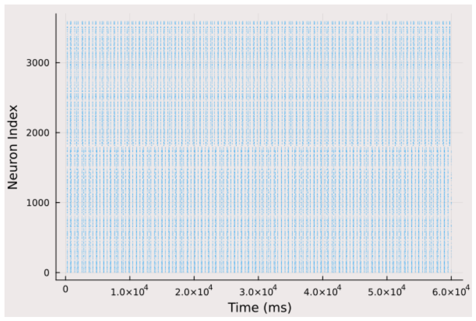
Two common models of cortical spiking networks are the, Potjan's and Diesmon [5] model and the Brunel model [6], both of these models exist within a fluctuation driven regime. When each of these respective network models are simulated, observed spike times are typically appear to be poisson distrubited psuedo random spike times. By design these models make it unlikely that fine grained recognizable repeating patterns also occur. The Potjan's model can be used to make data points seperable. However, new data sets prominently capture replayed states, and previously collected spike trains may, too, have latent and unpublished states of replay. The limited recordings from limited species may have biased previous recordings in a way that underrepresented the prevalence of replay.



One labelled sample of the NMNIST data set was fed into a Potjans connectome SNN, and the readout from the whole network was recorded.



The adjacency matrix from a random connectome approximately balanced E/I network with 4 different populations EI, EE, IE, II.



The spike trains read out after stimulating the above basic balanced connectome, after all of the neurons in the network were stimulated with direct current over the duration of the simulation recording.

References

[1] O. Awile, P. Kumbhar, et al., "Modernizing the neuron simulator for sustainability, portability, and performance," *Frontiers Neuroinformatics*, vol. 16, 2022.

[2] Y. Lu, "Spiking neural networks," *Published Electronically Https://github.-com/astupidbear/spikingneuralnetworks.jl*, 2020.

[3] B. J. Arthur, C. M. Kim, S. Chen, S. Preibisch, and R. Darshan, "A scalable implementation of the recursive least-squares algorithm for training spiking neural networks," *Biorxiv*, pp. 2022–9, 2022.

[4] B. Illing, W. Gerstner, and J. Brea, "Biologically plausible deep learning —but how far can we go with shallow networks?," *Neural Networks*, vol. 118, pp. 90–101, 2019.

[5] T. C. Potjans, and M. Diesmann, "The cell-type specific cortical microcircuit: relating structure and activity in a full-scale spiking network model," *Cerebral Cortex*, vol. 24, no. 3, pp. 785–806, 2014.

[6] N. Brunel, "Hebbian learning of context in recurrent neural networks," *Neural Computation*, vol. 8, no. 8, pp. 1677–1710, 1996.