UNIVERSITY OF THE PHILIPPINES

Master of Science in Physics

Reinier Xander A. Ramos

Dynamics of a nonlinear cellular automata model of biological neurons and its applications to young and aged neurons

ENDORSED FOR SCHEDULING OF ORAL DEFENSE:

Thesis Adviser:

20230518

Johnrob Y. Bantang, Ph.D.

National Institute of Physics

University of the Philippines Diliman

Date of Submission:
June 2023

Thesis classification:
F

This thesis is available to the public.

ABSTRACT

Dynamics of a nonlinear cellular automata model of biological neurons and its applications to young and aged neurons

Reinier Xander A. Ramos
University of the Philippines, 2023

Adviser:
Johnrob Y. Bantang, Ph.D.

Hodgkin-Huxley (HH) and other computational neuronal models are adequate methods in describing the dynamics of a single neuron. However, the HH model consist of four ordinary differential equations (ODEs) to solve for the neuronal response. Hence, simulating a large network of HH neurons would require very powerful computing devices. A model based on cellular automata (CA) has been recently made possible resulting to a simplistic way of simulating many neurons without requiring large computational resource. In this thesis, the dynamics of neuronal patch network involving an approximated nonlinear activation function are numerically explored. On one hand, cobweb and fixed point analyses reveal the evolution of the response over time of a single neuron within its localized environment. On the other hand, the bifurcation diagrams show steady-state behavior of the neuronal patch on the global scale. Additionally, changing the initial spatial conditions reveal different spatiotemporal patterns emerging from the neuronal patch activity. The collective dynamics examined within this thesis provide insight and computational groundwork of a simplified large-scale neuronal network for future research and development. As a proof of concept, we applied the modeling paradigm to young and aged neuronal cell data and infer about their possible dynamical properties.

PACS: 82.40.Bj [Oscillations, chaos, and bifurcations], 87.19.lj [Neuronal network dynamics], 87.19.ll [Models of single neurons and networks]

Table of Contents

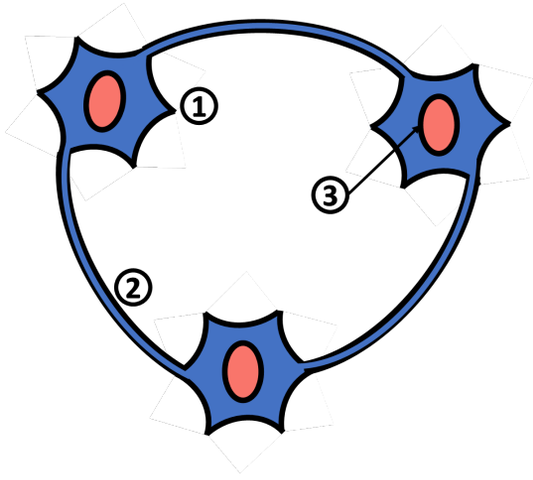
List of Figures and Tables

Chapter 1

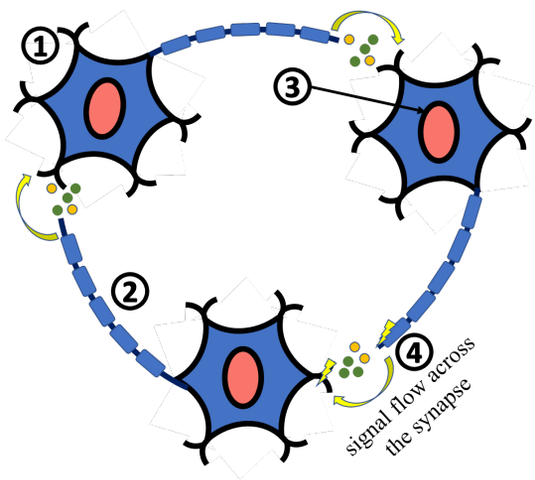
Neuronal Systems Modeling

In the early 19th century, when light microscopy was prominent, the nervous system was believed to be a single continuous network [1, 2]. This belief is called "reticular theory". Camillo Golgi is one of the major proponents of this theory where he showed intricate description of the dendrites and axons through his own black staining method. However, the theory was rivaled and refuted by neuron theory. The difference in the structure of the nervous system as proposed by reticular and neuron theory are shown in Figure 1.1. According to the neuron theory, the nervous system is composed of large number of units called neurons, and that the neural system is discontinuous by the existence of synapses in-between these neurons [3–5]. Santiago Ramon y Cajal first showed this discontinuity using the same staining method as Golgi. Both shared the 1906 Nobel Prize in Physiology or Medicine in recognition of their work on the structure of the nervous system [6].

Years later in 1952, Alan Hodgkin and Andrew Huxley provided the first mathematical model that inherently contains spiking mechanism describing the initiation and propagation of action potential in the neuron. They formulated a circuit model that is empirically consistent with the data as shown in Figure 1.2. The model is described by a set of four ordinary differential equations (ODEs) that characterizes the amount of ion concentration diffusing through the channels across the neuronal membrane during the



(a) reticular theory



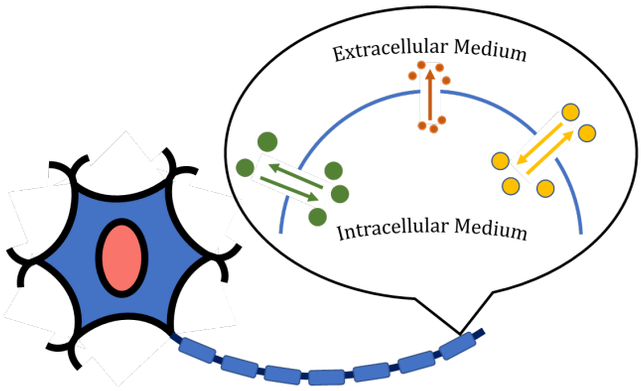
(b) neuron theory

Figure 1.1: *Reticular theory vs neuron theory.* In reticular theory, the nervous system is a single continuous system without contiguous parts. On the other hand, the neuron theory states that the nervous system is discontinuous and is composed of fundamental units called neurons. The neuron is partitioned to the following: 1) dendrite, receiver of signal; 2) axon, transmitter of signal; 3) soma, interpreter of signal; and 4) synapse, spatial junction for which electrochemical response occurs in-between neurons.

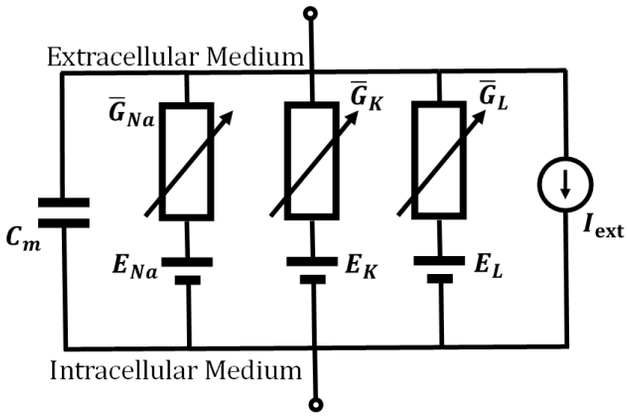
spiking period [7, 8]. Both shared the 1963 Nobel Prize in Physiology or Medicine, together with Sir John Eccles, for their discoveries concerning the

ionic mechanisms involved in excitation and inhibition in the peripheral and central portions of the nerve cell membrane [9].

Further experiments in neuroscience led to the discovery that neurons exhibit summation of the input signals to generate a corresponding output signal [10, 11]. This summation phenomenon is then taken into account in constructing models of interconnected neurons [12].



(a) HH diffusion model



(b) equivalent HH circuit

Figure 1.2: *HH model of diffusion and its equivalent circuit.* Hodgkin and Huxley described the neuronal activity to be dependent on the amount of ions diffusing through gated channels across the neuronal membrane. In the analogous circuit, the membrane acts as a capacitor C_{mem}, while the channels are modeled as conductances $\overline{G}_{\text{ion}}$ and reversal potential E_{ion} for ion $\in \{Na,\ K,\ \text{leak}\}$. The injected current or stimulus is shown as I_{ext}.

1.1 Complexity of the Nervous System

The neuron theory proved the fundamentality of the neuron to the nervous system. The basic function of the neuron is to receive, to interpret, and to transmit electric signals. The components of the neuron [13, 14] are partitioned in Figure 1.1(b) as follows.

1. The shorter branches known as dendrites acting as the receiver;

2. The soma containing the nucleus interprets the signal; and

3. The longer branches called axons acting as the transmitter.

Under the reticular theory shown in Figure 1.1(a), dendrites and axons also exist but are connected continuously as if in one singular network. But as observed from experiments, there exists a space in-between neurons called the synapse [15, 16]. The signal transmission occurs in these junctions as ions are released and/or absorbed. The presence of the synapses nullified the reticular theory and paved the way to the formalization of the neuron theory. In this latter theory, the nervous system is composed of a large number of individual units called neurons which are discontinuous (with some exceptions) and communicate only via synapses.

Later, it was discovered that some groups of axons and dendrites spike synchronously when a stimulus is injected into one of them [17]. These bundles of axons and dendrites (without nucleus) are known as neuroglia (glial cells). The glial cells help maintain homeostasis in the nervous system by holding neurons in place, insulating them, supplying oxygen and nutrients, and destroying pathogens [18–20]. A group of neuroglial cells and neuronal cells form the nervous tissue. Each type of nervous tissue has a specialized function. In the brain, the grey matter relays motor and sensory information, while the white matter is responsible for learning and long-range signal transmission. Other more common nervous tissues are the nerves that conduct short-range signal transmission, and ganglia which affects cognition and involuntary movements [21, 22].

At the organ-level, the nervous system has the brain, spinal cord, and sensory organs. The brain is the main organ of the nervous system, while the spinal cord is the main pathway of signal transmission in vertebrates, connecting the brain to almost every other part of the body. The brain is considered as one with the highest complexity and is responsible for the systematization of a lot of bodily functions such as movement, emotion, cognition, learning, memory, and consciousness [21, 22]. Until today, scientists are challenged to explain why and how the brain came to be a complex structure.

The above structural distinction and hierarchy of the nervous system varies from one literature to another. There is no clear classification as to whether neuroglia and ganglia are considered as tissues or otherwise [23]. These bundles contain axons (myelinated and/or unmyelinated), dendrites, and sometimes may contain nuclei. There is also no clear boundary in-between nerves and sensory organs. In this thesis, neurons considered are only those with dendrites and axons that communicate the input and output neuronal signals respectively. Furthermore, we refer to a collection of neurons as a neuronal tissue or neuronal network.

1.2 Model of a Spiking Neuron

Figure 1.3 shows the different spiking regimes during the generation of an action potential in the neuron [24]. Neurons can exhibit a brief rise in the voltage difference across their membrane. This voltage known as the action potential is recorded via probe electrodes into the neuron. An unstimulated neuron is said to be in a "resting" or "quiescent" state and maintains $\Phi_{\mathrm{Q}} \approx -70$ mV resting potential.

When subjected to a stimulus signal, the neuron may undergo different stages of electrochemical polarization. If the stimulus does not exceed the inherent threshold voltage (usually at $\Phi_{\mathrm{thr}} = \Phi_{\mathrm{Q}} + 15\ \mathrm{mV} = -55\ \mathrm{mV}$), the

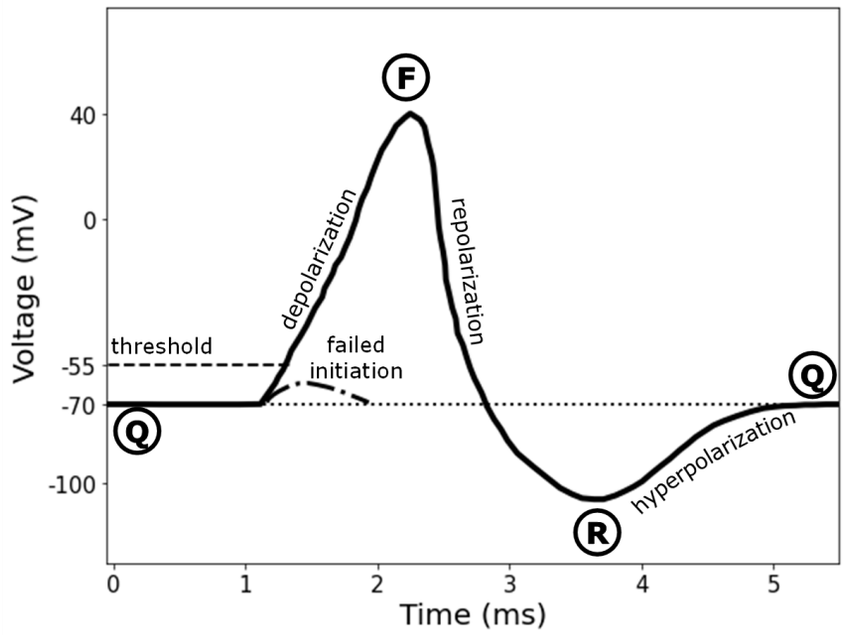


Figure 1.3: *Generation of an action potential in the neuron.* The neuron typically maintains a resting potential at the "Q"uiescent state. When the stimulus exceeds the threshold voltage, the neuron undergoes depolarization of its ion channels, peaking at the "F"iring state. The neuron immediately repolarizes reaching a potential below the resting potential and transitions to its "R"efractory state, where the neuron hyperpolarizes back to "Q". The potential curve is patterned from Gerstner et al. [8].

neuron will fail to initialize the spiking process. If it does exceed, ion channels across the membrane open leading to the inrush of positively charged ions in the process of depolarization. The neuron will then reach its "firing" or "spiking" state which reaches up to $\Phi_{\mathrm{F}} \approx 40$ mV. After which, repolarization stage starts where the voltage drops below the threshold voltage.

At the lowest voltage of the repolarization process, the neuron transitions to its "refractory" state, starting a period of recovery back to the resting potential. The hyperpolarization stage is when the potential gradually recovers back to the quiescent state. The neuronal states are summarized in Table 1.4.

Table 1.4: Various spiking regimes and corresponding states of a neuron during a generation of an action potential.

State	Symbol	Status of neuron
Firing	F	neuron is spiking or active
Refractory	R	neuron is undergoing hyperpolarization
Quiescent	Q	neuron is in the resting potential or inactive

Integrate-and-Fire Model

One of the earliest mathematical models that describes the neuronal spiking process is the integrate-and-fire (IAF) model first formulated by Louis Lapicque in 1907 [8, 25]. He relates the action potential $\Phi(t)$ generated over time to the input current according to

$$I(t) = C\frac{d\Phi(t)}{dt} \tag{1.1}$$

where C is the inherent membrane capacitance. The equation shows that the voltage increases indefinitely with the current until the cell is said to "fire". This means a reset of the voltage must be imposed in the model each time the potential exceeds the threshold. However, the reset property is not observed in biological neurons. The model has been modified in the succeeding years until 1952 when Alan Hodgkin and Andrew Huxley first presented experimental validation of the Hodgkin-Huxley model with the IAF model as its inspiration and basis.

1.3 The Hodgkin-Huxley Model

Hodgkin and Huxley modeled the neuron as a circuit shown in Figure 1.2. In this model, the neuronal membrane is treated to have voltage-gated ion channels that open or close depending on the amount of ion concentration on either side of the membrane [7, 8, 26]. The ion channels start as closed during the resting state and will open if the stimulating current exceeded

the threshold voltage, allowing ions to pass through. The channels will again close during the refractory period to maintain the voltage difference.

$$\begin{aligned} C\frac{d\Phi}{dt} &= \overline{G}_{Na}m^3h(E_{Na}-\Phi) + \overline{G}_K n^4(E_K-\Phi) \\ &\quad + \overline{G}_{\text{leak}}(E_{\text{leak}}-\Phi) + I_{\text{ext}}(t) \\ \frac{dm}{dt} &= \alpha_m(1-m) - \beta_m m \\ \frac{dn}{dt} &= \alpha_n(1-n) - \beta_n n \\ \frac{dh}{dt} &= \alpha_h(1-h) - \beta_h h \end{aligned} \tag{1.2}$$

Equation 1.2 showcases two main ions that diffuse during neuronal spiking: sodium for fast-activation channel, and potassium for slow-activation channel. The electrical conductances $\overline{G}_{\text{ion}}$, for ion $\in \{Na,\ K,\ \text{leak}\}$ are inherent to each ion channel, as well as the capacitance C to the neuronal membrane. The electrochemical gradients E_{ion} drive the flow of ions through the channel.

Table 1.5: Parameter constants based on Hodgkin and Huxley (1952) [7].

Variable	Parameter Name	Value
C	Membrane capacitance	1 μF/cm^2
$\overline{G}_{Na}$	Sodium conductance	120 mS/cm^2
$\overline{G}_K$	Potassium conductance	36 mS/cm^2
$\overline{G}_{\text{leak}}$	Leakage conductance	0.3 mS/cm^2
E_{Na}	Sodium reversal potential	115 mV
E_K	Potassium reversal potential	-12 mV
E_{leak}	Leakage reversal potential	10.6 mV

The quantities m, n, h are the probabilities associated with sodium channel activation, sodium channel inactivation, and potassium channel activation, respectively. These probabilities are described by the Boltzmann trans-

port equations α_χ and β_χ, for $\chi \in \{m,\ n,\ h\}$, in the form of

$$\begin{aligned} \alpha_\chi(\Phi) &= \frac{\chi_\infty(\Phi)}{\tau_\chi} \\ \beta_\chi(\Phi) &= \frac{1 - \chi_\infty(\Phi)}{\tau_\chi} \end{aligned} \tag{1.3}$$

where $\chi_\infty(\Phi) = \lim_{t\to\infty} \chi(\Phi)$ are the steady-state values as a function of voltage and τ_χ are the time constants. The values for the constants were experimentally determined by Hodgkin and Huxley as they were probing electrodes on a squid giant axon [7].

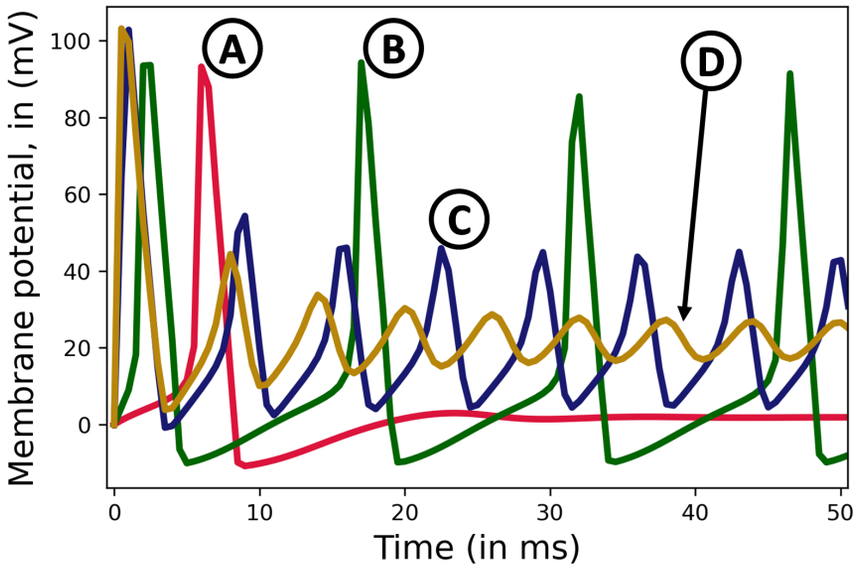
Solution to the HH Model

Equation 1.2 can be solved by Euler method or any other similar ODE solving algorithms. For a given constant input current, we plot the membrane potential over time as shown in Figure 1.6(a). At very low currents ($I_{\text{ext}} < 6.232\ \mu\text{A/cm}^2$), the neuron shows single to no spiking (corresponds to the red cross and red line) [27]. However, the frequency of spikes increases with the input current at ($6.232\ \mu\text{A/cm}^2 \leq I_{\text{ext}} \leq 155\ \mu\text{A/cm}^2$). Beyond $155\ \mu\text{A/cm}^2 < I_{\text{ext}}$, the spiking frequency slowly decreases until there is no more spiking event.

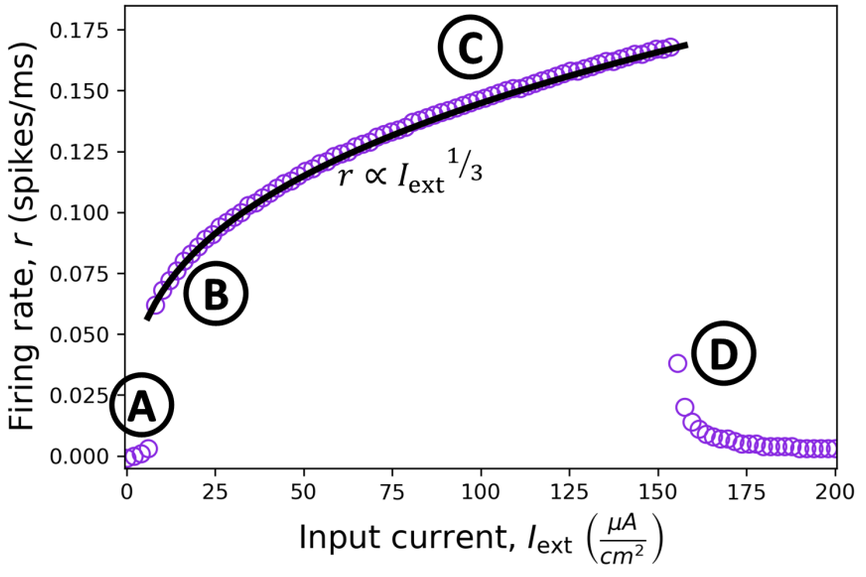
From the membrane potential curve, we count the number of spikes per ms and measure the firing rate $r = f(I_{\text{ext}})$ as shown in Figure 1.6(b). This is the activation or response function of the HH neuron defined as $f = a(\text{input})$.

1.4 Single-Neuron Dynamics

In their experiments involving the squid giant axon, Hodgkin and Huxley varied the stimulating current injected and recorded the frequency of spikes generated over time as the current is increased [7]. They distinguished between three classes of neuronal excitation based on how the spiking behavior changes with respect to the input current. Each class of excitability has its corresponding activation function profile.



(a) HH representative solution



(b) HH activation function

Figure 1.6: *HH solution and activation function.* Hodgkin-Huxley representative membrane potential curves over time (top) sampled from the activation function (bottom). The color corresponds to different input current I_{ext}-values in μA/cm^2: (A) 2.50, (B) 10.0, (C)100.0, (D) 150.0. Image patterned from Pang (2014) [26].

1.4.1 Excitability Classes

Three main types of excitability were outlined in the original model [7, 28]. The potential curves and their corresponding activation function are shown in Figure 1.7.

Class 1 Tonic-spiking. The neuron exhibits sustained spiking activity regardless of the strength of the stimulating current. The frequency of spikes is proportional to the strength of stimulating current, hence providing a continuous activation function.

Class 2 Phasic-spiking. The neuron undergoes two phase transitions as the stimulating current is increased. At very low currents, the neuron produces only one spike. At a critical current value, the neuron will spike only in the first few milliseconds of the stimulation period.

Class 3 Single-spiking. The neuron only has one phase transition: it produces only a single spike once the stimulating current is high enough.

On one hand, the activation function of HH Class 1 is continuous due to its sustained spiking. On the other hand, the other two HH Classes are discontinuous at the phase transition boundary. The HH model and its derivatives (i.e. Morris-Lecar model [8, 28]) were able to simulate the activation profile of these three neuronal excitability classes.

1.4.2 Approximation to the Activation Function

In general, the specific values of the activation function may vary depending on the type of neuron that is subjected for experimental observation. However, what remain the same across every experiment are the properties of the activation profile which are summarized by Ramos et al. [29] as follows.

1. Two thresholds (a minimum and a maximum) in the input are present indicating that neurons fire only when stimulated by an input current between these two thresholds. We respectively assign for these the thresholds a_0 and a_1, the minimum and maximum, respectively.

2. The firing rate monotonically increases whenever the input signal (i.e. current) corresponds to an a_{in}-value between a_0 and a_1; the firing rate is zero otherwise.
3. A maximum threshold in the output is present limiting the firing rate values for the entire range of a_{in}. We assign this as a_2.

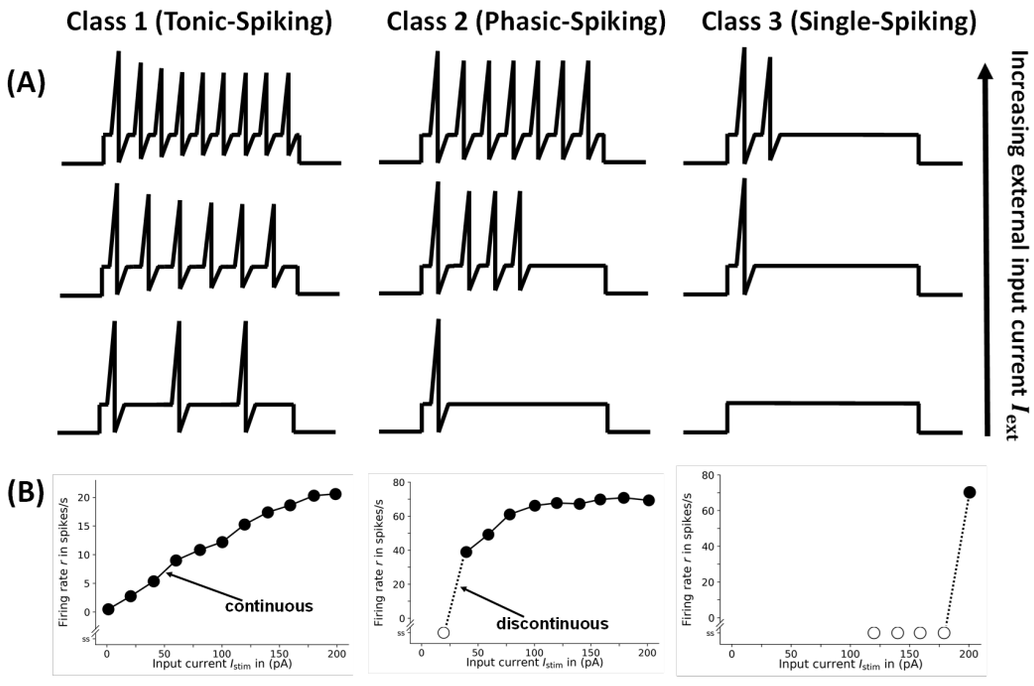


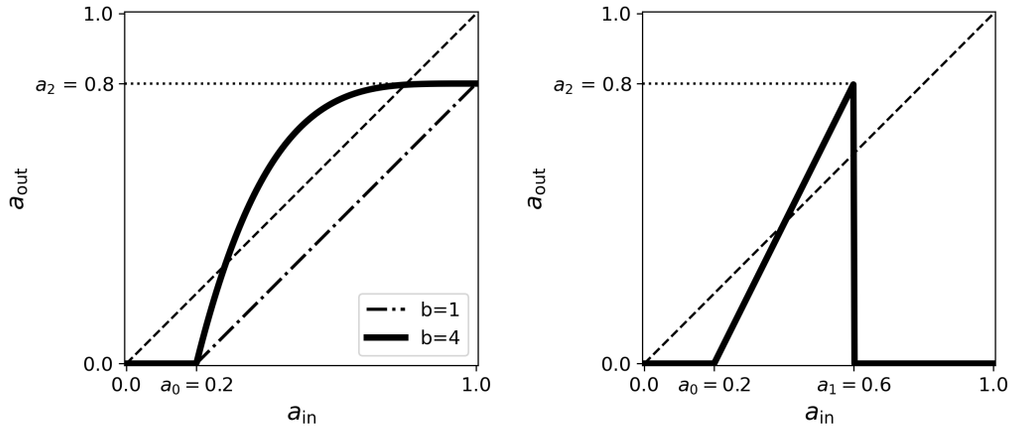
Figure 1.7: *Excitability classes and corresponding activation function in HH model.* Top panel: Neuronal response of experiments with (from top to bottom) decreasing stimulus current. The response is only recorded during the stimulation period. Bottom panel: Corresponding activation function profile for each class of excitability. Open circles represent instances where single spike (ss) occurred during the full stimulation period. Closed circles represent instances where activity is sustained throughout the stimulation period. Potential curves and activation function profiles are patterned from experiments done by Prescott et al. [28].

We model in Figure 1.8(a) the response that incorporates the thresholds as a nonlinear activation function given by

$$a_{\text{out}} = \begin{cases} 0 & 0 < a_{\text{in}} < a_0 \\ a_2\left(1 - \left(1 - \frac{a_{\text{in}} - a_0}{1 - a_0}\right)^b\right) & a_{\text{in}} \geq a_0 \end{cases} \tag{1.4}$$

where b is the nonlinearity parameter. In the limit as $b \to 1$, the response becomes linear and simplified as follows.

$$a_{\text{out}} = \begin{cases} 0 & a_{\text{in}} < a_0 \\ \frac{a_2(a_{\text{in}} - a_0)}{a_1 - a_0} & a_{\text{in}} \in [a_0, a_1] \\ 0 & a_{\text{in}} > a_1 \end{cases} \tag{1.5}$$



(a) full approximation

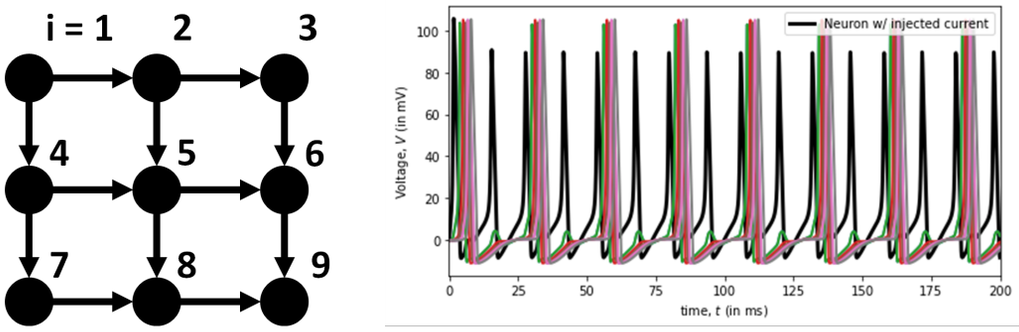
(b) linear approximation ($b \to 1^+$)

Figure 1.8: *Nonlinear and linear approximation to the HH activation function.* The thresholds a_0 and a_1 corresponds to the minimum and maximum input thresholds respectively. In the nonlinear case, we set $a_1 = 1$ to reduce the number of parameters when performing exhaustive search. The parameter a_2 represents the maximum output threshold while b determines the nonlinearity of the function. In the limit as $b \to 1$, the response becomes linear.

1.5 Network of HH Neurons

Although a powerful tool, the HH model is limited by the solvability of coupled ODEs to simulate a large number of interconnected neurons. Each

set of ODEs are also coupled to all the other neurons connected, hence the need for additional coupling terms in Equation 1.2. One such method is to build the directed network of interconnected HH neurons [26, 27] as shown in Figure 1.9(a). Some even have used analog circuits and microchips to build the neuronal network and presented promising efficiency [30]. But, applying these methods to larger number of neurons require expensive computational resources and longer time to solve [29].



(a) HH network

(b) response of HH network

Figure 1.9: *HH network and response.* Directed network of HH neurons (a) as implemented in [26, 27], and the corresponding response (b) of each neuron. An external input is constant to the node $i = 1$ (black voltage curve). Colored voltage curves correspond to the other eight neurons in the network. The response of the neurons at the end of the directed network are the most delayed as compared to the stimulating input current. Image patterned from Pang (2014) [26].

Today, it remains a challenge in the field of neuroscience to explain how a human brain works and how complex behaviors such as learning, memory, and consciousness are expressed arising from connected simple neurons [31]. The advent of technology pushed the HH model to its limits, relying on greater computational resources. This thesis aims to provide a more efficient method in describing large ($N \sim 10^6$) neural systems while maintaining the properties exhibited by HH neurons. In the next chapter, we present an improved version of a neuronal cellular automata model that provides a faster and more efficient method of simulating interconnected neurons.

Chapter 2

Neuronal Cellular Automata

"Is it possible for a machine to reproduce itself?" This was the question imposed by John von Neumann in the late 1940's in his journey of finding self-replicating systems. He dreamt of a robot that can automatically replicate and evaluate itself, hence the term automata [32, 33]. In the late 1950's, he worked with Stanislaw Ulam who suggested to treat the liquid to be composed of discrete units, where the motion of each unit is calculated based on the behavior of neighboring units [34]. The result was the first system of a cellular automaton (CA).

The method of discretizing a system into units arranged in a regular lattice or network was not much popular until John Conway published his Game of Life in the 1970's [35]. The Game of Life (GoL) is a two-state CA with such simple rules but resulted to a wide variety of dynamical behaviors when left running. Numerical experiments on the Game of Life showed that the simple rules can be used to build logic gates, "living" configurations, and configurations that copies itself and destroys the original.

After the popularity of GoL, other physical systems such as weather, cloud formation, molecular gas, and even population migration networks, have been studied using CA models. As computational technology advances, CA models can simulate much larger physical systems. CA models are pushed to the limits of the computational power one has [36].

2.1 Components of Cellular Automata

The five-tuple that defines a two-dimensional (2D) rectangular cellular automata [37] used in this work is given by:

$$\text{CA} = \{\ \mathcal{S}, \mathcal{C}, \mathcal{L}, \mathcal{N}, \mathcal{R}\ \} \tag{2.1}$$

where

$\mathcal{S} =$ $\{0, .., S-1\}$, the set of identifiers for the unique state of a "cell" where S is the number of states a "cell" can be.

$\mathcal{C} =$ $\{c = (i,j) |\ i \in [1, 2, \ldots, \text{W}],\ j \in [1, 2, \ldots, \text{Z}]$ s.t. $\text{W} \cdot \text{Z} = N\}$, the set of identifiers of each CA "cell" where N is the total number of cells . If the cells are arranged in a square lattice, then $W = Z = L$, where L is the lattice size. Additionally, (i, j) easily refers to the location of the cell. We then write $s_c = s_{i,j} \in \mathcal{S}$ as the state of the cell $c \in \mathcal{C}$.

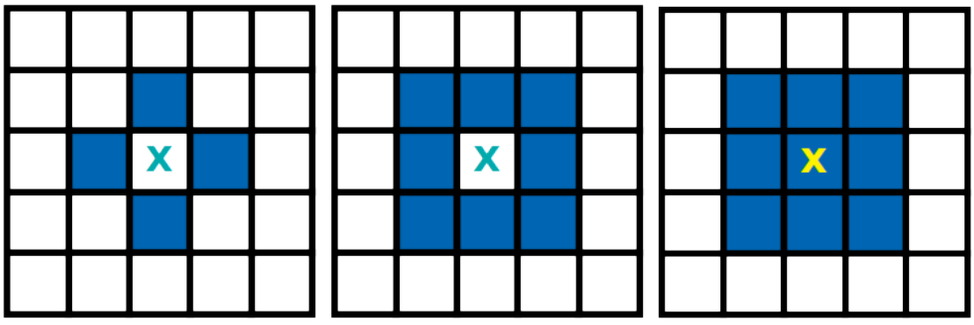
$\mathcal{L} =$ defines the lattice neighborhood which is generally a mapping $f : \mathcal{C} \to \mathcal{C}^M$ where M is the total number of neighboring cells in the lattice. Any given cell c is mapped to another tuple of cell identifiers: $L_{i,j} = \{(i-1, j-1),\ (i-1, j),\ (i-1, j+1), \ldots,\ (i, j)\}$. We then say that in such lattice $\mathcal{L}$, the cells in $L_{i,j}$ is a neighbor of $c = (i, j)$. In general, M is not a constant and may be dependent on either c or $\mathcal{L}$, or both.

$\mathcal{N} =$ $\mathcal{S}^M$, the set of state neighborhood. Thus $N_c = N_{i,j} \in \mathcal{N}$ such that each N_c is in the form of the M-tuple: $\{s_{i-1,j-1}, s_{i-1,j}, s_{i-1,j+1}, \ldots, s_{i,j}\}$.

$\mathcal{R} =$ defines the set of rules implemented in the CA with $g : s_{i,j} | \mathcal{N} \to \mathcal{S}$ as the mapping of any neighborhood state N_c to a new state $s'_{i,j}$ of the cell c. At the next time step, $s'_{i,j}$ replaces the original $s_{i,j}$.

For simplicity, we only consider the arrangement of cells $\mathcal{C}$ in a square lattice. We define various neighborhood conditions as shown in Figure 2.1. The von Neumann neighborhood is defined as the connections to the primary directions (i.e. North, East, West, South). The Moore neighborhood is a von Neumann neighborhood extended to the secondary directions. We also

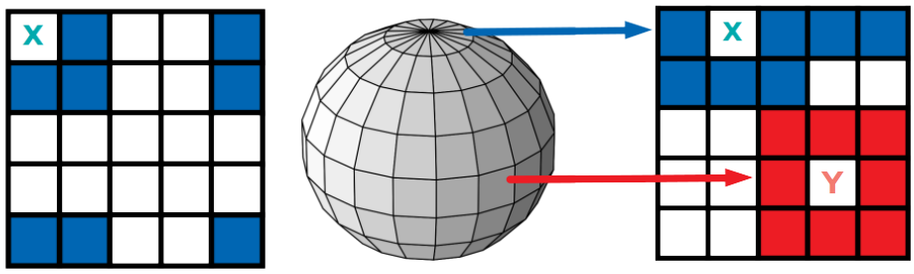
distinguish between totalistic and outer-totalistic where the center cell is part of the neighborhood or otherwise.



(a) von Neumann (b) outer-totalistic Moore (c) totalistic Moore

Figure 2.1: *Common lattice neighborhood conditions used in CA modeling.* In the von Neumann neighborhood, the cell 'X' is connected to its four adjacent neighbors in the primary directions (N, E, W, S), while Moore neighborhood is extended to the secondary directions. Under the totalistic type of neighborhood, the cell 'X' itself is considered part of the neighborhood. Otherwise, the neighborhood an outer-totalistic type.

Figure 2.2 shows various lattice boundary conditions to be imposed as part of the neighborhood. The lattice boundary conditions describe the connections of the cells located at the edge of the lattice. Under the toroidal boundary condition, cells on the leftmost column (topmost row) are connected to the rightmost column (bottommost row). This creates a wrap-around effect making the lattice looks like infinite on all sides. In the spherical boundary condition, the surface of the sphere is mapped to a square lattice via Mercator projection [38–40]. This leaves a regular Moore neighborhood on the "equatorial" cells while leaving the "polar" cells to be fully connected. The spherical boundary condition allows us to model epithelial neurons — cells that are located at the surface of the brain.



(a) toroidal (b) spherical projected

Figure 2.2: *Common lattice boundary conditions used in CA modeling.* In toroidal boundary condition, cells at the edges are connected to the other side to create a wrap-around effect across the lattice. In spherical boundary condition, the sphere's surface is mapped into a square lattice via Mercator projection.

Conway's Game of Life

The GoL CA is played on a square lattice of length L with totalistic Moore neighborhood and toroidal boundary conditions. The total number of cells in the lattice is $N = L \times L$. Each cell is assigned to be "alive" or "dead" state. Further CA specifications are detailed below.

$\mathcal{S}$ $= \{0, 1\}$ where 1 is assigned to be "alive" and 0 for the "dead" state.

$\mathcal{C}$ $= \{(1,1),\ (1,2),\ \ldots,\ (1,L),\ (2,1),\ (2,2),\ \ldots, (2,L),\ \ldots,\ (L,L)\}$ are the identifiers for each cell in the square lattice $L \times L$.

$\mathcal{L}$ $= f(c) \leftarrow \Big[L_c = \{(i-1, j-1),\ (i-1, j),\ (i-1, j+1),\ (i, j-1),\ (i, j+1),\ (i+1, j-1),\ (i+1, j),\ (i+1, j+1)\}\Big]$ as a mapping for outer-totalistic Moore neighborhood with toroidal boundary condition.

$\mathcal{N}$ $= \{00000000, 00000001, \ldots, 11111111\}$ such that the representation $N_c \in \mathcal{N}$ is equivalent to $N_c = \{s_{i-1,j-1},\ s_{i-1,j},\ s_{i-1,j+1},\ s_{i,j-1},\ s_{i,j+1},\ s_{i+1,j-1},\ s_{i+1,j},\ s_{i+1,j+1}\}$.

$\mathcal{R}$ $= \{g(s_{i,j}) = \delta_{s,0}\delta_{3,\Lambda(c)} + \delta_{s,1}\left(\delta_{2,\Lambda(c)} + \delta_{3,\Lambda(c)}\right)\}$ at the next time step where

$$\Lambda(c) = \sum_{c' \in L_c} s'_c \tag{2.2}$$

The update rule simply states that any "dead" cell with exactly three "alive" neighbors, that is $\Lambda(c) = 3$, becomes "alive" at the next time step. While, any "alive" cell dies when either $\Lambda(c) > 3$ or $\Lambda(c) < 2$.

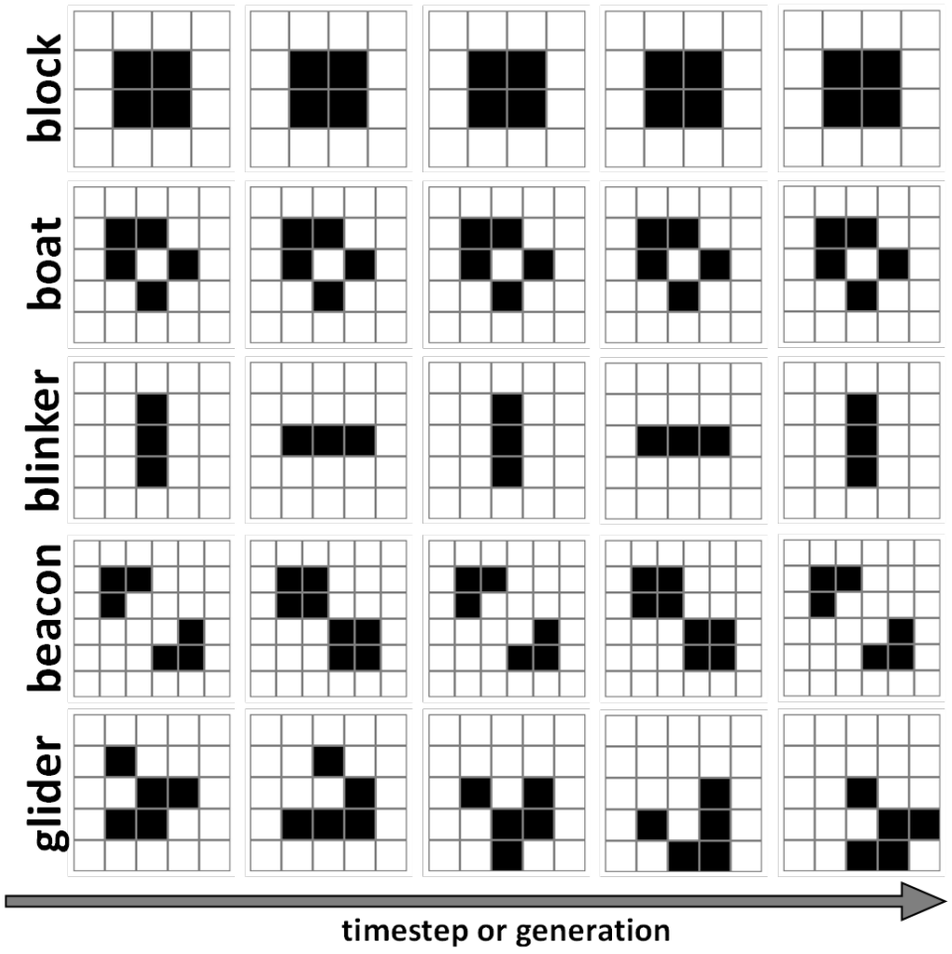


Figure 2.3: *Patterns emerging from various initial spatial configuration within the Game of Life.* The first two (block, boat) are still-lifes. The blinker and the beacon are oscillators with period two, while the glider has a period of four with the whole pattern shifted spatially as if it "glides" across the lattice.

When played, different patterns evolve from various initial spatial configurations as shown in Figure 2.3. Some patterns does not change completely and are called "still-lifes" (i.e. block, boat). Other patterns evolve as oscillators which cycles through a series of patterns per generation. Blinker

and beacon are oscillators with period-two. Patterns such as spaceships (not shown) and gliders tend to move or creep across the lattice. Still some other patterns either grows infinitely, or dies over longer periods of time.

2.2 Discrete Neuronal CA

Recall from Section 1.2 the different stages of neuronal spiking. We discretize each stage as follows:

1. Quiescent (resting) state
2. Firing (action potential is generated) state
3. Refractory state.

Thus we construct the set of states $\mathcal{S} = \{R, Q, F\} = \{0, 1, 2\}$ corresponding to the discrete spiking stages. These are the states of the Brian's brain CA (BBCA), first studied by Brian Silverman in 1996 [41, 42]. The BBCA is typically simulated in a square lattice with toroidal boundary condition and outer-totalistic Moore neighborhood conditions. Further CA specifications are presented below.

$\mathcal{S}$ $= \{R, Q, F\} = \{0, 1, 2\}$ where R=Refractory, Q=Quiescent, F=Firing.

$\mathcal{C}$ $= \{(1,1),\ (1,2),\ \ldots,\ (1,L),\ (2,1),\ (2,2),\ \ldots,(2,L),\ \ldots,\ (L,L)\}$ are the identifiers for each cell in the square lattice $L \times L$.

$\mathcal{L}$ $= f(c) \leftarrow \Big[L_c = \{(i-1, j-1),\ (i-1, j),\ (i-1, j+1),\ (i, j-1),\ (i, j+1),\ (i+1, j-1),\ (i+1, j),\ (i+1, j+1)\}\Big]$ as a mapping for outer-totalistic Moore neighborhood with toroidal boundary condition.

$\mathcal{N}$ $= \{00000000, 00000001, \ldots, 22222222\}$ such that the representation $N_c \in \mathcal{N}$ is equivalent to $N_c = \{s_{i-1,j-1},\ s_{i-1,j},\ s_{i-1,j+1},\ s_{i,j-1},\ s_{i,j+1},\ s_{i+1,j-1},\ s_{i+1,j},\ s_{i+1,j+1}\}$.

$\mathcal{R}$ $= \{g(s_{\mu,\nu}) = (s_C + 1)\delta_{R,s} + (s_c + 1)\delta_{Q,s}\delta_{2,\Lambda(c)} + (s_c - 2)\delta_{F,s}\}$ at the next time step where

$$\Lambda(c) = \sum_{c' \in L_c} \delta_{F, s_{c'}} \tag{2.3}$$

Starting from a random uniform distribution of states, BBCA typically stabilizes with gliders colliding with each other as shown in Figure 2.4. The directional movement of the gliders seemingly simulate signal transmission across the automaton, hence the resemblance to a neuronal network.

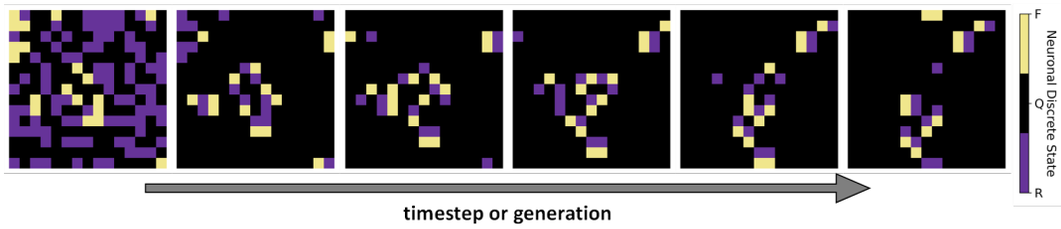


Figure 2.4: *Patterns emerging from random initial spatial configuration within the Brian's Brain CA.* The emergence of directional movement of the gliders is analogous to signal transmission across a neuronal patch, hence the resemblance to the brain.

2.3 Continuous Neuronal CA with Linear Activation Function

In our previous works [43, 44], we proposed a continuous state neuronal CA that utilizes the linear activation function described in Section 1.4.2. The neuronal CA explored is in a square lattice with toroidal boundary condition and totalistic Moore neighborhood. Let a denote the net activity of a neuron has at a specific time. As an input a_{in} to the cell, the net activity can be interpreted as the stimulating current, ion concentration, or the probability of firing of its neighbors. As an output a_{out} of the cell, the net activity can be interpreted as the membrane potential, or the probability to fire at the next time step. Further CA specifications are detailed below.

$\mathcal{S} = \{a \in \mathbb{R}, 0 \leq a \leq 1\}$ where a represents net activity of the neuron at that time step.

$\mathcal{C} = \{(1,1),\ (1,2),\ \ldots,\ (1,L),\ (2,1),\ (2,2),\ \ldots,(2,L),\ \ldots,\ (L,L)\}$ are the identifiers for each cell in the square lattice $L \times L$.

$\mathcal{L} = f(c) \leftarrow \Big[L_c = \{(i-1, j-1),\ (i-1, j),\ (i-1, j+1),\ (i, j-1),\ (i, j+1),\ (i+1, j-1),\ (i+1, j),\ (i+1, j+1),\ (i, j)\}\Big]$ as a mapping for totalistic Moore neighborhood with toroidal boundary condition.

$\mathcal{N} = \{000000000, \ldots, 111111111\}$ such that the representation $N_c \in \mathcal{N}$ is equivalent to $N_c = \{a_{i-1,j-1},\ a_{i-1,j},\ a_{i-1,j+1},\ a_{i,j-1},\ a_{i,j+1},\ a_{i+1,j-1},\ a_{i+1,j},\ a_{i+1,j+1},\ a_{i,j}\}$.

$\mathcal{R} = \{g(a_{i,j}) = \Lambda(c)\}$ at the next time step with

$$\Lambda(c) = \frac{1}{M} \sum_{c' \in L_c} a_{c'} \tag{2.4}$$

is the average a-states in the neighborhood of $a_{i,j}$ and $M = 9$.

The rule R is implemented for a duration until the average activity $\langle a \rangle_{CA}$ of the CA reaches its steady-state — when the average change in the neuronal network activity is zero. By performing exhaustive search on the parameter

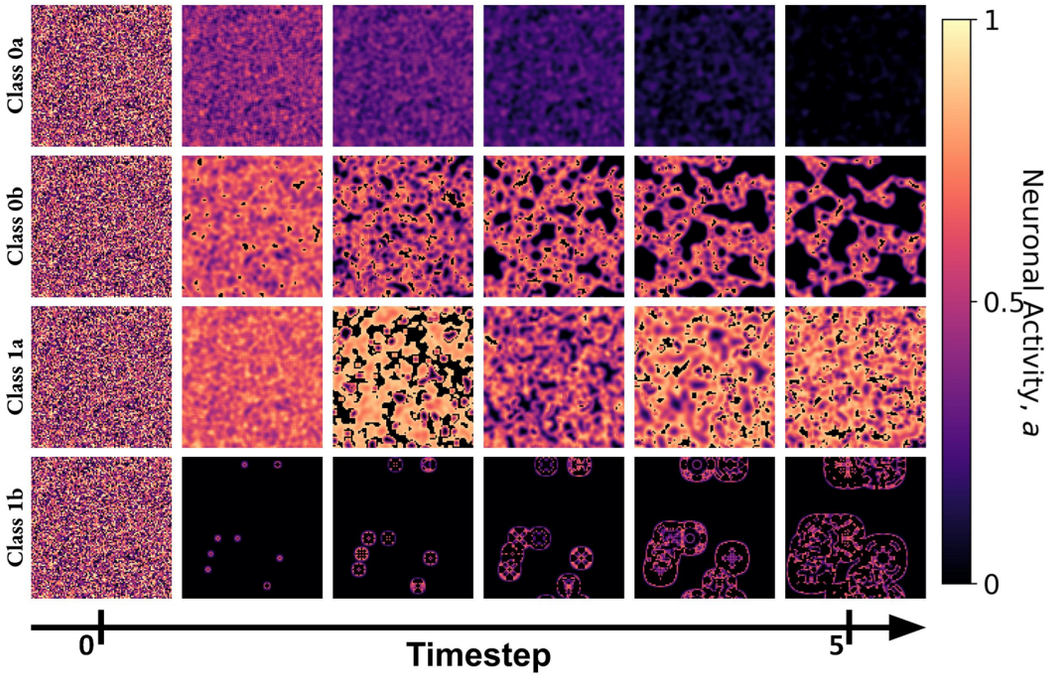


Figure 2.5: *Spatiotemporal patterns observed from each subclasses of Class 0 and Class 1 NCA.* From top row to bottom row: Class 0a, Class 0b, Class 1a, Class 1b.

space (a_0, a_1, a_2), we classified the steady-state behavior as follows.

1. Class 0: Quiescent Steady-State:
   (a) Fast-decay ; (b) Slow-decay
2. Class 1: Spiking Steady-State:
   (a) With random patterns ; (b) With exploding patterns
3. Class 2: Oscillating Steady-State

Figure 2.5 shows the spatiotemporal patterns generated from each subclasses in Class 0 and Class 1. For Class 0, we distinguish between fast-decay and slow-decay of the net neuronal activity towards a quiescent steady-state. For Class 1, the net neuronal activity is sustained in all time steps but with either random patterns or exploding patterns. To obtain the patch or network dynamics, we plot the neuronal patch average $\langle a \rangle_{CA}$ over time in Figure 2.6.

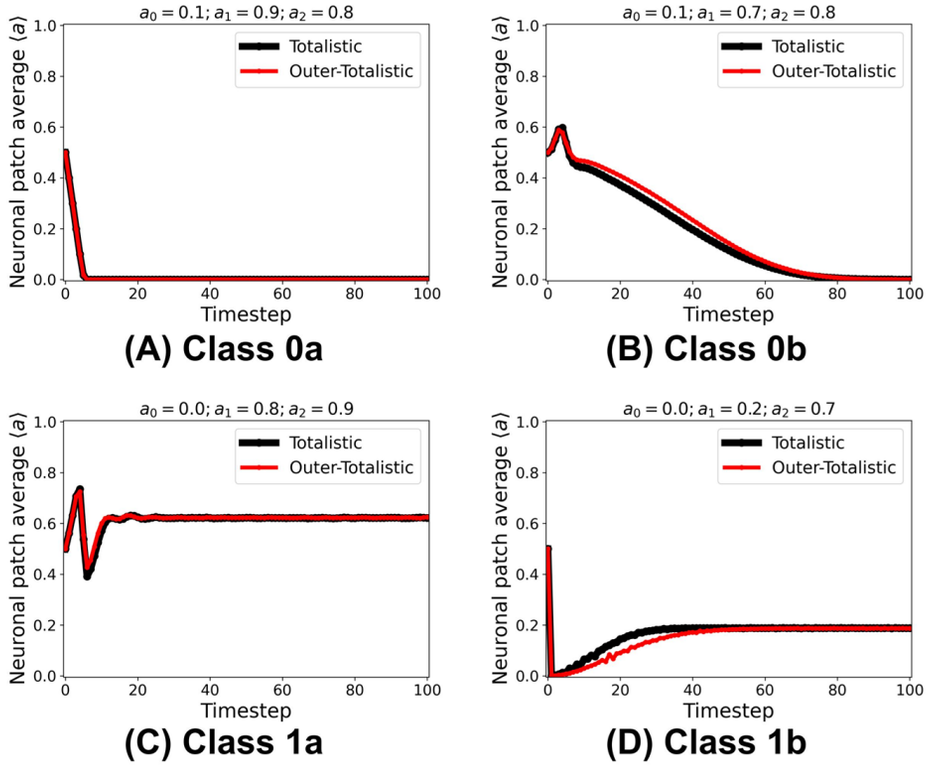
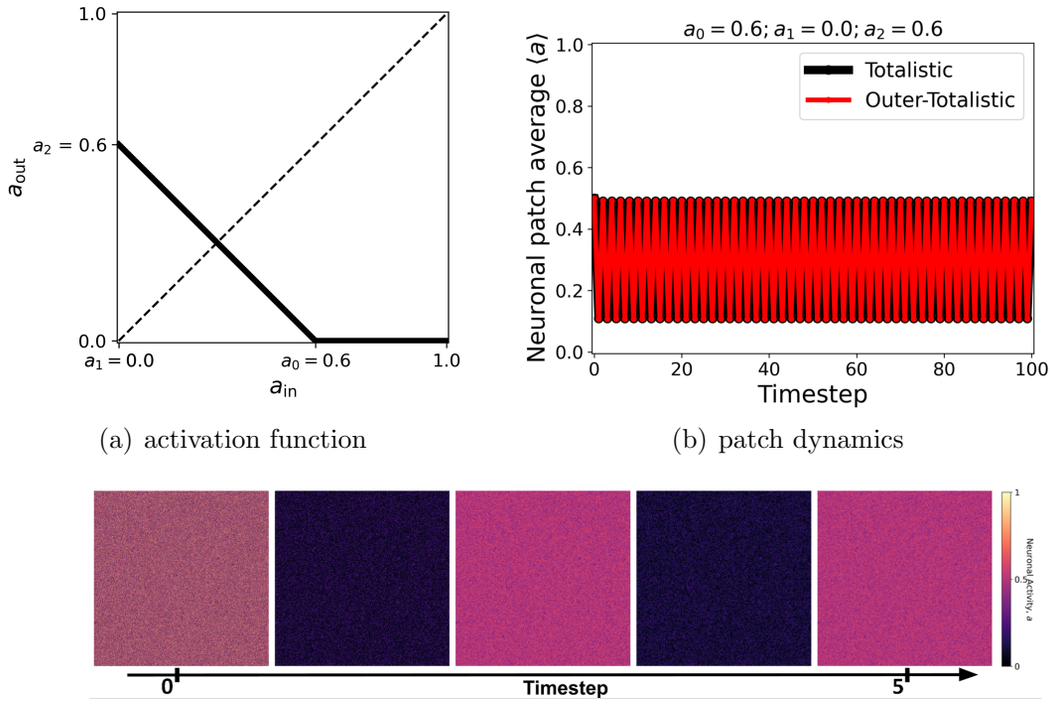


Figure 2.6: *Representative totalistic vs outer-totalistic dynamics of each subclass of linear NCA.* The dynamics is obtained by recording the average activity of the patch $\langle a \rangle_{CA}$ over time.

Class 2 NCA is only observed when the activation function is inversely linear, that is, the function has negative slope. This occurs when $a_1 < a_0$ as shown in Figure 2.7(a). The oscillation is easily noticeable from the dynamics and the generated spatiotemporal pattern is a blinker of period two.



(a) activation function

(b) patch dynamics

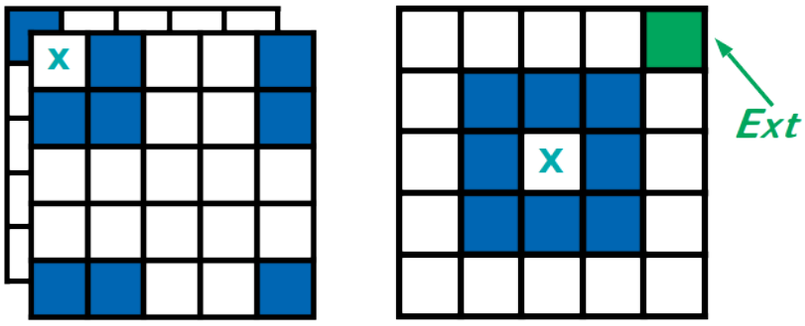
(c) spatiotemporal patterns

Figure 2.7: *Dynamics of Class 2 (Oscillating Steady-State) NCA.* The oscillating steady-state behavior is only achieved when the activation function is negatively slope, that is, $a_1 < a_0$. The spatiotemporal pattern generated is a blinker with a period two.

2.4 Extended Neighborhood and Boundary Conditions

Besides those that are shown in Figures 2.1 and 2.2, we define other lattice neighborhood and boundary conditions that are more suited for real neural systems. For instance, in the grey and white matter, the bundles of neurons

may form sheaths of neuronal networks that are overlaid on top of each other similar to a graphene structure. A possible model for such system is a multi-layered CA as shown in Figure 2.8(a), where the cell is connected to all the neurons along the axis of overlay in addition to its Moore neighborhood on the cell's layer.



(a) two-layered (b) with external input I_{ext}

Figure 2.8: *Extended lattice boundary conditions in NCA.* In a multi-layered CA, the neuron is connected to all the neurons along the axis of overlay, as well as its Moore neighborhood on its layer. To introduce external current or stimulus to the CA, we can define a fraction of the cells in the CA to be injected with external input $a_{\text{ext}} = 1.0$.

Let Z be the total number of layers where a cell's location is extended to the k-th layer in addition to its (i, j) location within the lattice. Hence, we extend the three elements of $\mathcal{A}$ as follows:

$\mathcal{C}$ $= \{(1,1,1),\ (1,2,1),\ \ldots,\ (1,L,1),\ (2,1,1),\ (2,2,1),\ \ldots,(2,L,1),\ \ldots,$ $(L,L,1),\ \ldots,\ (L,L,Z)\}$ are the identifiers for each cell in the square lattice $L \times L$ and Z is the total number of layers.

$\mathcal{L}$ $= f(c) \leftarrow \Big[L_c = \{(i-1,j-1,k),\ (i-1,j,k),\ (i-1,j+1,k),\ (i,j-1,k),$ $(i,j+1,k),\ (i+1,j-1,k),\ (i+1,j,k),\ (i+1,j+1,k),\ (i,j,k),\ (i,j,k+$ $1)\}\Big]$ as a mapping for totalistic Moore neighborhood with toroidal boundary condition in a two-layered $(Z = 2)$ lattice.

$\mathcal{N}$ $= \{0000000000, \ldots, 1111111111\}$ such that the representation $N_c \in \mathcal{N}$

is equivalent to $N_c = \{a_{i-1,j-1,k},\ a_{i-1,j,k},\ a_{i-1,j+1,k},\ a_{i,j-1,k},\ a_{i,j+1,k},\ a_{i+1,j-1,k},\ a_{i+1,j,k},\ a_{i+1,j+1,k},\ a_{i,j,k},\ a_{i,j,k+1}\}$.

Another neighborhood condition we can define is the external input condition in which a fraction ξ of the cells in $\mathcal{C}$ in the CA are constantly injected with an external input $a_{\text{ext}} = 1.0$. This does not change any of the five-tuple that defines our automaton only that some elements of $\mathcal{N}$ are fixed to 1. This conforms to experiments involving probing current and/or stimulus on the neurons through electrodes, where the activity for those neurons is high and constant.

2.5 Computational Time Complexity

In this section, we quantify the efficiency of the cellular automata method as compared to solving the exact ODEs in Equation 1.2 using typical ODE solvers. We considered a square network of HH neurons and used the following ODE solvers for the network.

1. Forward Euler
2. Fourth-order Runge-Kutta (RK4)
3. Livermore Solver for ODEs Algorithm (LSODA)

For each solver (including the cellular automata method), we recorded the average of three runs as the computational time T it takes to finish a simulation. We find that the relationship between T and the neural system size N is best described [29] by the following fitting function

$$T = \eta\,(N - \zeta)^{\gamma} \tag{2.5}$$

with $\eta,\ \zeta,\ \gamma$ as the fitting parameters. Of these parameters, γ determines the computational complexity since $T \sim \mathcal{O}(N^{\gamma})$ for large N-values.

Figure 2.9 compares the simulation time between these different algorithms. On one hand, solving a network of HH neurons using forward Euler method yields quadratic time complexity ($T \sim \mathcal{O}(N^2)$). Other ODE solvers

such as RK4 and LSODA provide better accuracy at the cost of higher T yet shows consistent complexity $\gamma \approx 2$. On the other hand, our NCA model presents a linear time complexity ($T \sim \mathcal{O}(N)$) showing that our model provides much faster computational time, especially for much larger system size N, while maintaining the integrity of the neuronal spiking behavior.

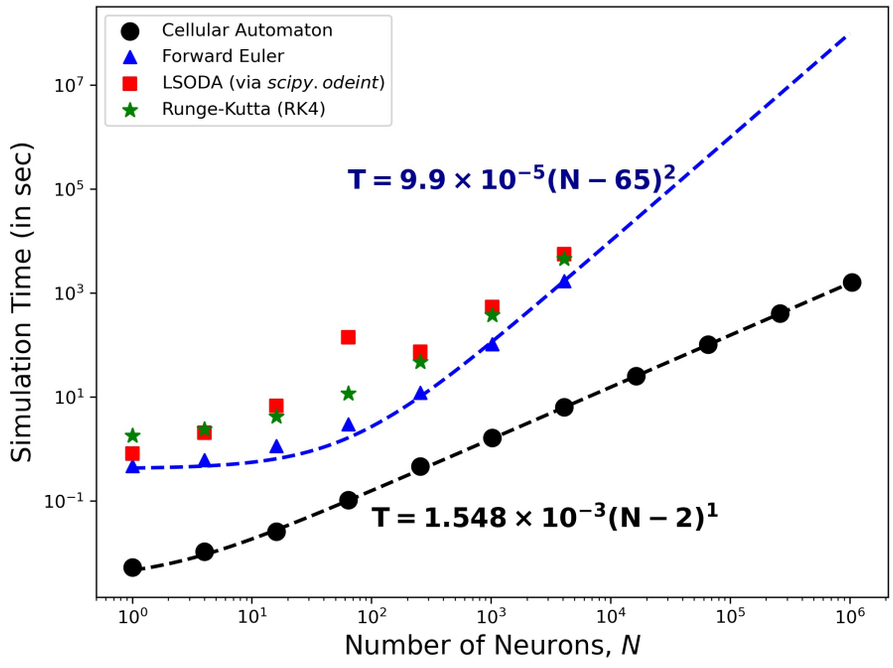


Figure 2.9: *Time complexity of the neuronal model comparing different algorithms for solving the HH dynamics.* The forward Euler method (marked ▲) shows quadratic order in time ($\sim O(N^2)$) for which other ODE solvers follow. On the other hand, the CA model (marked •) presents a linear complexity time ($\sim O(N)$) indicating the efficiency of using CA model in simulating neuronal patch dynamics.

Additionally, the simulation time values T for $N > 4,096$ were not recorded for any of the ODE solvers. This is because running simulations for large size causes the computational device to exceed the available memory capacity. Thus, our NCA model shows significant advantage in both simulation time and memory requirement to analyze the behavior of neural systems. Moreover, the algorithm for the NCA model can be straightforwardly par-

allelized and be implemented on a GPU to amplify the neuronal population without much increasing the simulation time.

Other studies have also implemented other methods and/or algorithms with HH as its basis for modeling neuronal population dynamics. The study by Ananthanarayanan et al. [30] utilizes the Blue Gene/P supercomputer to implement cortical simulations of 10^9 neurons of a cat. The supercomputer comes with $147,456$ CPUs and 144 TB of main memory which yields efficiency of roughly $\sim 6 \times 10^3$ neurons/CPU and ~ 144 KB/neuron. Meanwhile, our NCA model is powered by 1 CPU and 16 GB of memory (RAM) which can easily handle a lattice with $N \sim 10^6$ with memory to spare for other calculations. For storage memory purposes, our NCA model exhibits an efficiency of up to 4×10^6 neurons/CPU and ~ 4 bytes/neuron. With this, all CA simulations in the next chapters are done in a $1,000 \times 1,000$ ($N = 10^6$) square lattice without the loss of generality.

Chapter 3

Bifurcation Theory

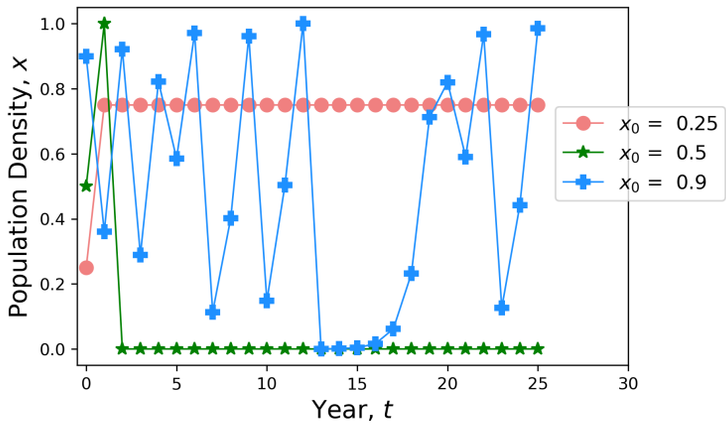
So far, we have discussed two fields utilized in this thesis: the neuron theory in Chapter 1, and cellular automata theory in Chapter 2. This chapter introduces the bifurcation theory used to explore the dynamics of the neuronal spiking behavior over the parameter space. The bifurcation theory provides various analyses that determines the stability of the mapping function used in the system.

3.1 Logistic Map

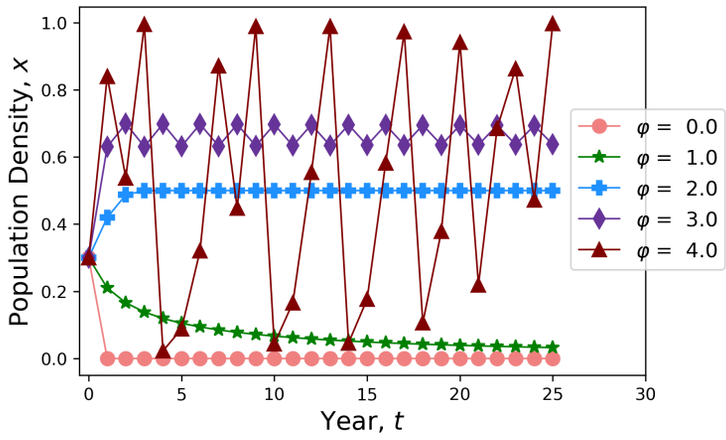
The closest well-known return map to our system is the logistic map. To demonstrate the power of bifurcation theory, let's consider as an example a population of rabbits as an isolated system P_{rab}. The population growth per year can be modeled [45, 46] by the logistic equation

$$x_{t+1} = \varphi x_t(1 - x_t) \tag{3.1}$$

where x_t is the population at year t, and φ is a constant growth rate. The factor $(1 - x_t)$ as a constraint to the population so that the steady-state does not explode to infinity. Figure 3.1 shows the population dynamics for varying initial population x_0 and varying growth rate φ. Some parameter values, i.e. at $(x_0,\ \varphi) = (0.25,\ 4)$, lead to constant population steady-state while some



(a) Varying x_0, $\varphi = 4$



(b) Varying φ, $x_0 = 0.3$

Figure 3.1: *Dynamics of logistic equation.* Shown are the population curves for varying initial population x_0 or varying growth rate. The steady-state dynamics is evident for $t > 5$.

lead to oscillating steady-state, i.e. at $(x_0,\ \varphi) = (0.3,\ 3.0)$. Others have random steady-state such as $(x_0,\ \varphi) = (0.9,\ 4)$.

3.2 State Space

The state of the system P_{rab} can be specified by (x_t, x_{t+1}). Plotting x_t vs x_{t+1} reveals the state space of the logistic equation. The resulting curve

in the state space is referred to as the input-output mapping, in this case the logistic map, as shown in Figure 3.2. A mapping characterized by a single hump is called a unimodal map, which means there exist a single local maximum for the mapping [45–47].

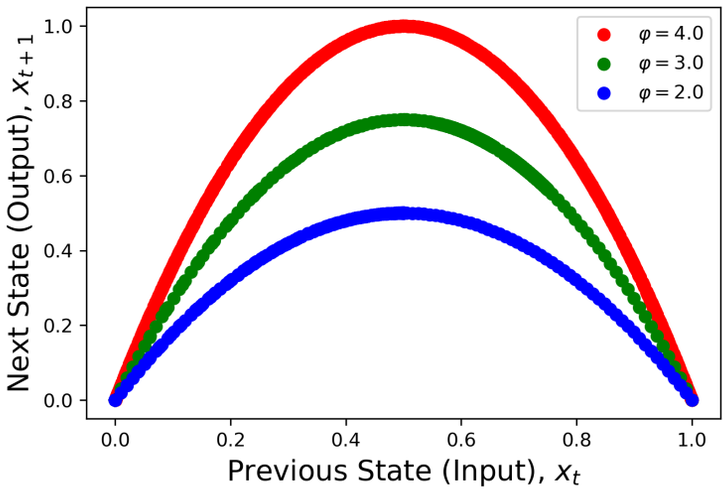


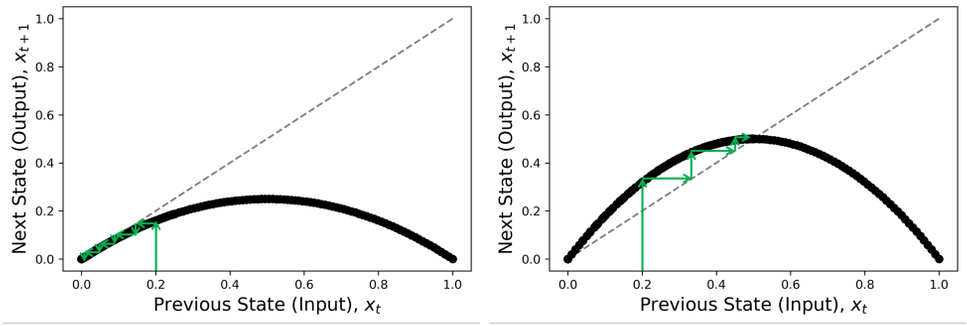
Figure 3.2: *State space of logistic equation.* The resulting curve is considered an input-output mapping of the function, hence the logistic map.

3.2.1 Cobweb Diagram

From the input-output map, we construct a cobweb diagram [29, 48, 49] on a plane $(X, Y) = (x_{\text{in}}, x_{\text{out}})$ as follows.

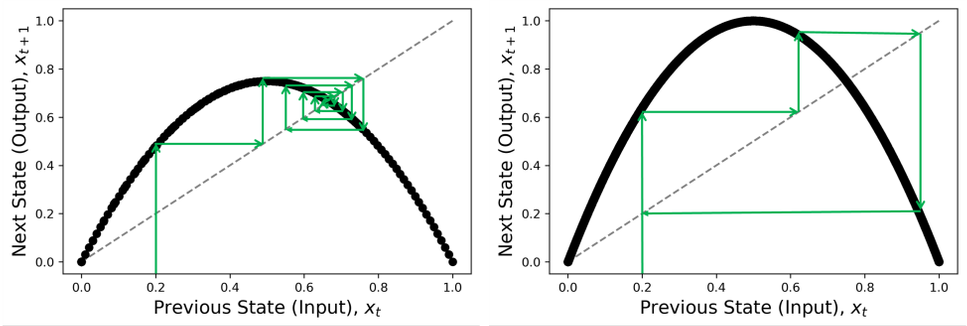
1. Given a chosen starting point $(X_{\text{start}}, Y_{\text{start}}) = (a_{\text{start}}, 0)$, we trace a vertical line from it to $(x_{\text{start}}, f(x_{\text{start}}))$.
2. We trace a horizontal line from $(x_{\text{start}}, f(x_{\text{start}}))$ until it crosses the diagonal (dashed line) with the equation $x_{\text{out}} = x_{\text{in}}$. This value becomes the new starting point, such that $(x_{\text{start}}, y_{\text{start}}) = (f(x_{\text{start}}), f(x_{\text{start}}))$.
3. Repeat steps 1 and 2 until x settles to an equilibrium point or an equilibrium trajectory.

Figure 3.3 shows the cobweb trajectories with the same initial state $x_0 = 0.2$ for varying growth rate φ. The cobweb diagrams more easily reveals the steady-state of the system for any given initial state x_0 and parameter φ just by tracing the trajectory. At $\varphi = 1.0$, the logistic map lies below the diagonal such that any initial state will reach a zero steady-state. At $\varphi = 2.0$, the maximum of the logistic map coincides with the center of the diagonal ($x_{\text{out}} = x_{\text{in}} = 0.5$). In this case, any initial state will stabilize at the maximum. We define an intersection point to be any point on the map that intersects with the diagonal such that $x_{t+1} = x_t$.



(a) $\varphi = 1.0$

(b) $\varphi = 2.0$



(c) $\varphi = 3.0$

(d) $\varphi = 4.0$

Figure 3.3: *Representative cobweb diagrams of logistic map for the same initial state* $x_0 = 0.2$. The gray dashed line represents the diagonal $x_{\text{out}} = x_{\text{in}}$. The green line represents the cobweb trajectory on the state space. The steady-state is obtained by finding if the trajectory settles to a point on the diagonal, or if the trajectory repeats itself for which it said to be an oscillating steady-state.

At $\varphi = 3.0$, the maximum is above the diagonal such that the intersection point between the map and the diagonal is greater than 0.5. Any initial state will steady-state at the intersection point but for a longer duration of the trajectory. When the maximum is located at the edge of the (X, Y) plane, in this case at $x_{\text{out}} = 1.0$, all trajectories will reach an oscillating steady-state.

3.2.2 Fixed Points and Stability

The points where the mapping function intersects with the diagonal are called the fixed points or the equilibrium of the system. Hence all intersection points are fixed points of the system. When the cobweb trajectory moves towards and ends at a fixed point, the fixed point is said to be stable. If the trajectory moves away from a fixed point, then we have an unstable equilibrium.

If the trajectory happened to be periodic or oscillating, then we call the trajectory a limit cycle. A limit cycle that converges into the same set of periodic states is a stable limit cycle. If the limit cycle diverges into infinity, then we have an unstable limit cycle.

For the logistic map, we have the following stability analysis:
$\varphi = 1$**:** one stable fixed point at $(0, 0)$.
$\varphi = 2$**:** one stable fixed point at $(0.5, 0.5)$; one unstable fixed point at $(0, 0)$.
$\varphi = 3$**:** one stable fixed point at $(0.\overline{6}, 0.\overline{6})$; one unstable fixed point at $(0, 0)$.
$\varphi = 4$**:** one stable limit cycle at $(0.75, 0.75)$; one unstable fixed point at $(0, 0)$.

3.3 Phase Space

The phase space or the parameter space provides insights regarding phase transitions that the system experiences when the system parameters are varied [50]. To obtain the phase space, we obtain the average $\langle x \rangle_{t \to \infty}$ points for which the system has reach its steady-state. Usually, these are state values x_t from the last t-steps of the simulation. We refer to the average as the

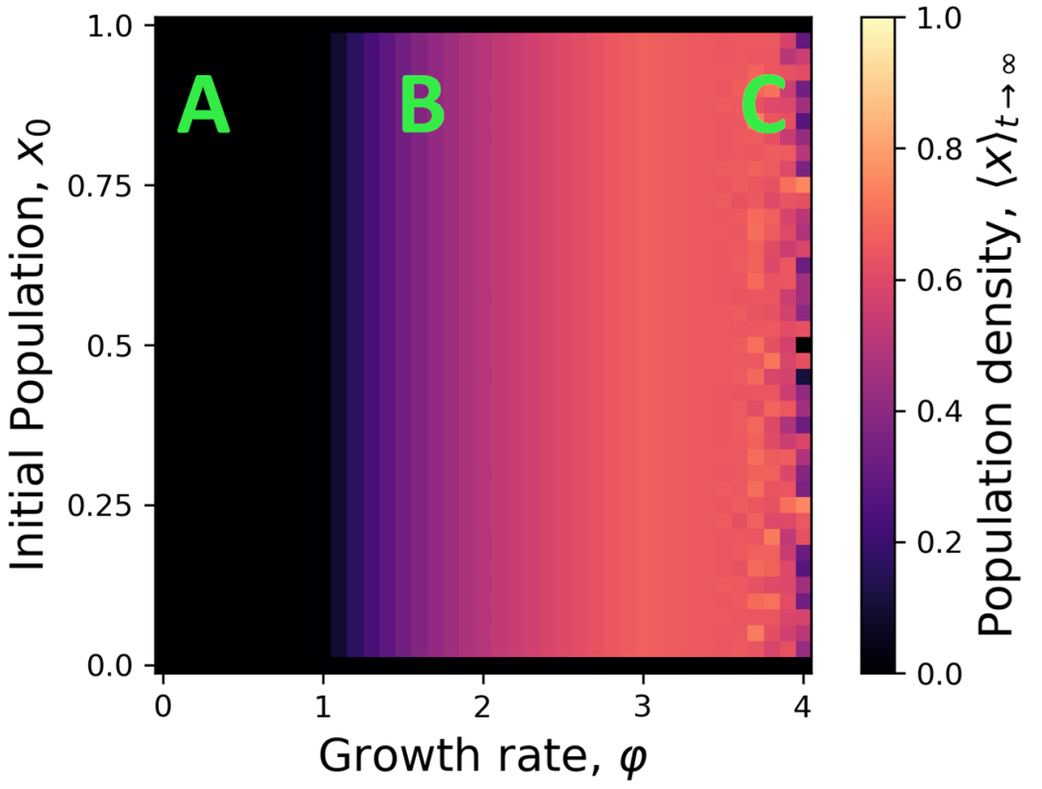


Figure 3.4: *Phase space of logistic equation.* The population density is taken as the average steady-state values for the last t-steps. Three phases are observed: A) zero steady-state, B) nonzero steady-state, and C) random steady-state.

final state of the system. We run through each possible combination of the system parameters to obtain the phase space Figure 3.4.

When the system has growth rate $\varphi \in [0, 1]$, the final state is always zero regardless of the initial state x_0. Between $\varphi = 1$ and $\varphi \approx 3.5$, the system transitions to a nonzero final state that is increasing with the growth rate for almost all values of x_0. Beyond $\varphi \approx 3.5$, the system undergoes another phase transition to a random final state independent of the initial state x_0.

3.4 Bifurcation Diagram

For another perspective of the dynamics of the system, we construct the bifurcation diagram and plot the steady-state values at the last t-steps versus a system parameter. When phase transitions are observed, then the system parameter is the bifurcation parameter [51].

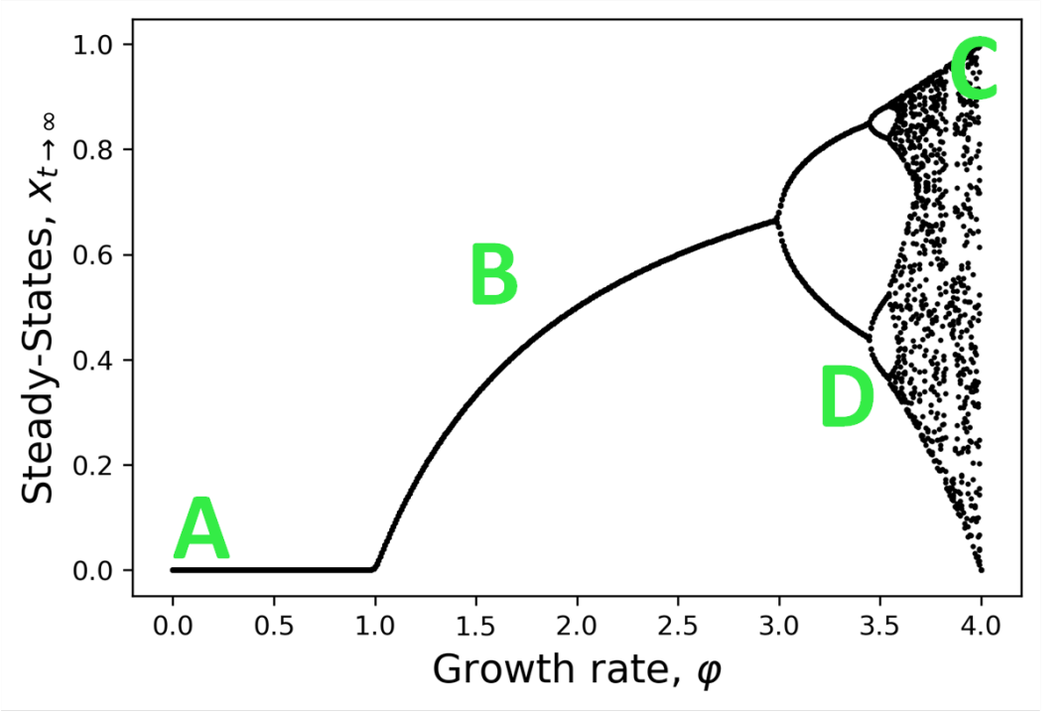


Figure 3.5: *Bifurcation diagram of logistic equation.* Four steady-state behaviors are observed: A) zero steady-state, B) nonzero steady-state, C) chaotic steady-state with islands of stability, and D) oscillating steady-state with period-doubling.

For the logistic map, the bifurcation parameter is the growth rate φ and the bifurcation diagram is shown in Figure 3.5. The zero steady-state at $\varphi \in [0,\ 1]$ and the nonzero steady-state at $\varphi \in (1,\ 3.0)$ are still observable. However, we now observe oscillating steady-state and period-doubling in the region $\varphi \in [3,\ 3.5]$ which is not visible previously in the phase space. This is called the period-doubling bifurcation. Furthermore, in the region beyond

$\varphi \approx 3.5$, chaotic steady-state is evident, but there exist islands of stability where periodicity is restored.

Bifurcation in Hodgkin-Huxley Model

The ODEs presented in the Hodgkin-Huxley model (see Equation 1.2 in Section 1.3) are transcendental functions making the bifurcation and steady-state analysis to be difficult. In Ashwin et al. [52], the HH ODEs are reduced into two coupled equations for easier analysis of the dynamics. The Φ-I state space is plotted in Figure 3.6 and each point is colored depending on the steady-state of the HH system at that input current.

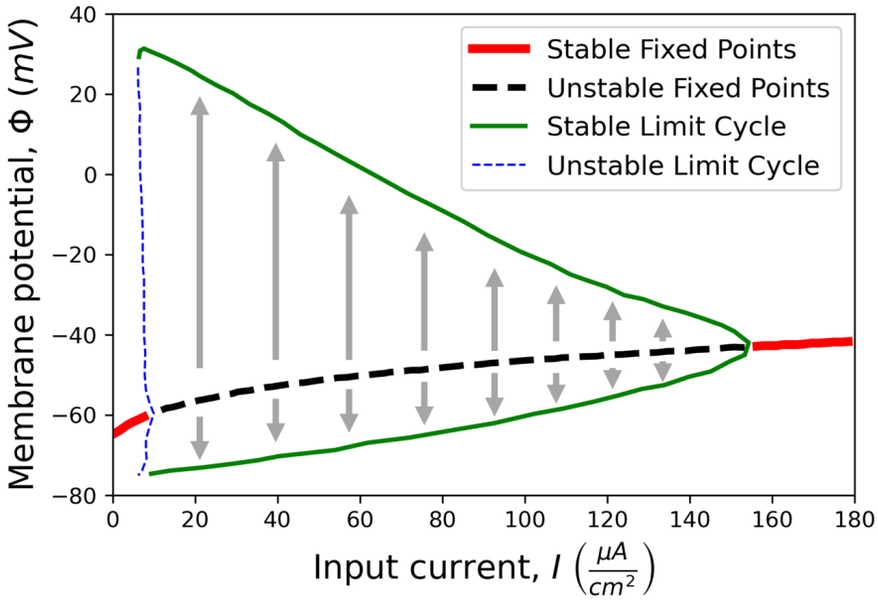


Figure 3.6: *State space and bifurcation diagram of reduced Hodgkin-Huxley model.* The line of stable fixed points corresponds to Class 3 HH or single-spiking excitability. The stable limit cycle represents Class 1 HH neurons with tonic-spiking, while the unstable limit cycle involves Class 2 HH neurons with phasic-spiking. Any neuron with a membrane potential above (below) the unstable fixed point line will settle to the top (bottom) half of the stable limit cycle, as indicated by the gray arrows). A fixed point that transitions to a limit cycle is called a Hopf bifurcation. Image patterned from Ashwin et al. [52].

The red line of points corresponds to stable fixed point in which the HH model tends towards the Q-state or zero firing rate ($r = 0$). On the other hand the black line indicates the line of unstable fixed points. The trajectory of the HH system moves away from these points and towards the limit cycle (green and blue line). Any neuron with a membrane potential above (below) the unstable fixed point line will settle to the top (bottom) half of the stable limit cycle.

The green line corresponds to a stable limit cycle where the HH system oscillates between the low and the high Φ-value given the input current $6.232\ \mu\text{A/cm}^2 \leq I_{\text{ext}} \leq 155\ \mu\text{A/cm}^2$. Meanwhile, the blue line indicates an unstable limit cycle because the HH system visit every potential Φ-values given the input current $155\ \mu\text{A/cm}^2 < I_{\text{ext}}$.

A stable fixed point that branches into a limit cycle is called a Hopf bifurcation. On one hand, at $I_{\text{ext}} = 155\ \mu\text{A/cm}^2$, the Hopf bifurcation is stable because the fixed point branches into a stable limit cycle. On the other hand, the Hopf bifurcation is unstable at $I_{\text{ext}} = 6.232\ \mu\text{A/cm}^2$ since the stable fixed point branches into an unstable limit cycle. Furthermore, the steady-state behavior observed from the bifurcation diagram can be cross-referenced to the classes of neuronal excitation observed in Section 1.4.1.

Chapter 4

Nonlinear Neuronal CA

To build the nonlinear neuronal CA, we will implement the same CA specifications as that of the linear neuronal CA (see Section 2.3), only this time we utilize the nonlinear activation function detailed in Equation 4.1 as our interaction rule. Lastly, we explore the stability of the resulting dynamics of the nonlinear neuronal CA.

$$a_{\text{out}} = \begin{cases} 0 & 0 < a_{\text{in}} < a_0 \\ a_2\left(1 - \left(1 - \frac{a_{\text{in}} - a_0}{1 - a_0}\right)^b\right) & a_{\text{in}} \geq a_0 \end{cases} \tag{4.1}$$

4.1 CA Specifications

The neuronal CA explored is in a square lattice with toroidal boundary condition and totalistic Moore neighborhood. Let a denote the net activity of a neuron has at a specific time. As an input a_{in} to the cell, the net activity can be interpreted as the stimulating current, ion concentration, or the probability of firing of its neighbors. As an output a_{out} of the cell, the net activity can be interpreted as the membrane potential, or the probability to fire at the next time step.

$\mathcal{S} = \{a \in \mathbb{R}, 0 \leq a \leq 1\}$ where a represents net activity of the neuron at that time step.

$\mathcal{C} = \{(1,1),\ (1,2),\ \ldots,\ (1,L),\ (2,1),\ (2,2),\ \ldots, (2,L),\ \ldots,\ (L,L)\}$ are

the identifiers for each cell in the square lattice $L \times L$.

$\mathcal{L} = f(c) \leftarrow \Big[L_c = \{(i-1, j-1),\ (i-1, j),\ (i-1, j+1),\ (i, j-1),\ (i, j+1),\ (i+1, j-1),\ (i+1, j),\ (i+1, j+1),\ (i, j)\}\Big]$ as a mapping for totalistic Moore neighborhood with toroidal boundary condition.

$\mathcal{N} = \{000000000, \ldots, 111111111\}$ such that the representation $N_c \in \mathcal{N}$ is equivalent to $N_c = \{a_{i-1,j-1},\ a_{i-1,j},\ a_{i-1,j+1},\ a_{i,j-1},\ a_{i,j+1},\ a_{i+1,j-1},\ a_{i+1,j},\ a_{i+1,j+1},\ a_{i,j}\}$.

$\mathcal{R} = \{g(a_{i,j}) = \Lambda(c)\}$ at the next time step with

$$\Lambda(c) = \frac{1}{M} \sum_{c' \in L_c} a_{c'} \tag{4.2}$$

is the average a-states in the neighborhood of $a_{i,j}$ and $M = 9$.

4.2 Spatiotemporal Analysis

Recall from Section 1.4.2 that we set $a_1 = 1$ to reduce the number of parameters back to only three. By varying the parameters $(a_0,\ a_2,\ b)$, we explore the parameter space and classify the CA by its behavior. We also analyze the effects of varying the common and extended neighborhood and boundary conditions on the observed steady-state behavioral classification. The spatial configuration at $t = 0$ is taken from a uniform random distribution of a-values for each cell.

Classes of Steady-State Behavior

Figure 4.1 shows the steady-state dynamics of the nonlinear NCA. Two main classes of steady-state behavior are observed as follows.

1. Class 0: Quiescent Steady-State:
   (a) Fast-decay ; (b) Slow-decay
2. Class 1: Spiking Steady-State:
   (a) Low activation ; (b) High activation

Class 0 NCA results to a quiescent steady-state by either a fast- or slow-decay. However, with Class 1 NCA, the steady-state value is nonzero but with either low or high average activation. No oscillating steady-state were observed within the exhaustive search performed.

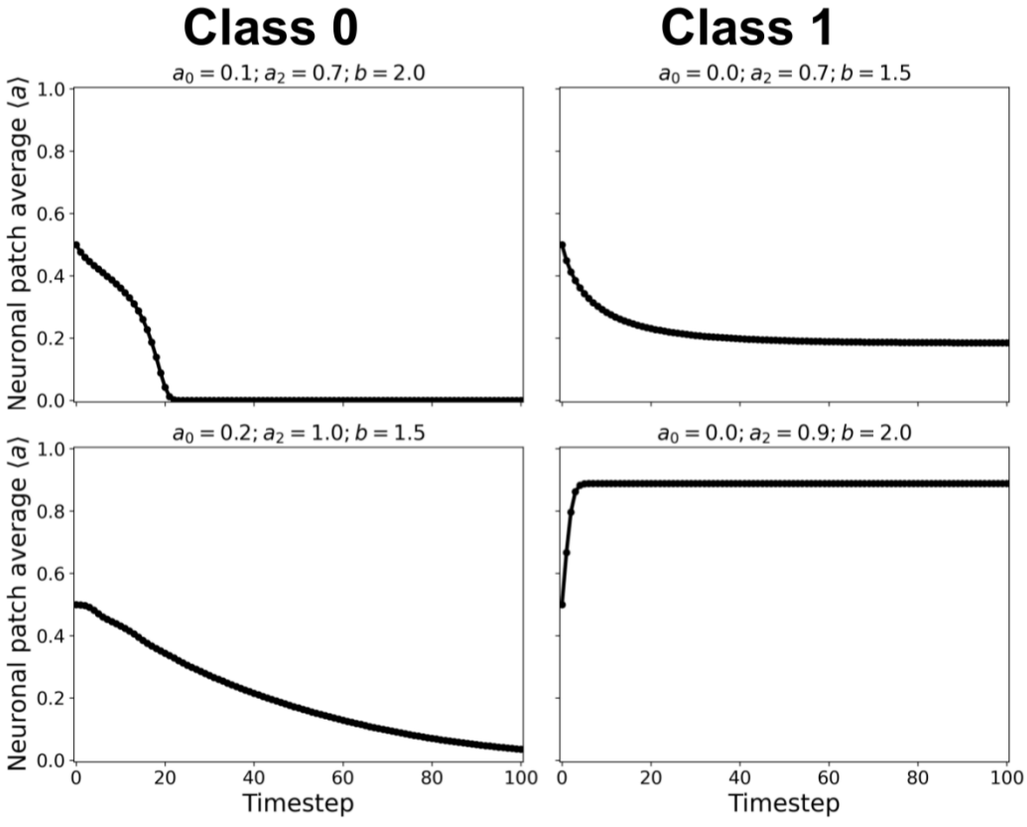


Figure 4.1: *Dynamics of nonlinear NCA.* Two major steady-state classes are observed. For Class 0, we observed a fast- and a slow-decay of activity towards zero steady-state. For Class 1, the activity is sustained throughout the duration with either low or high activation leading to spiking steady-state.

Effects of Neighborhood and Boundary Conditions

To investigate the effects of the lattice neighborhood and boundary conditions, we first consider a toroidal lattice and then vary between an outer-totalistic and a totalistic setting described in Figure 4.2. Although no change

in the classification is observed, the totalistic setting provides a much more closer model to biological neurons since a real neuron take into account its current state (i.e. ion concentration) as part of the activation function.

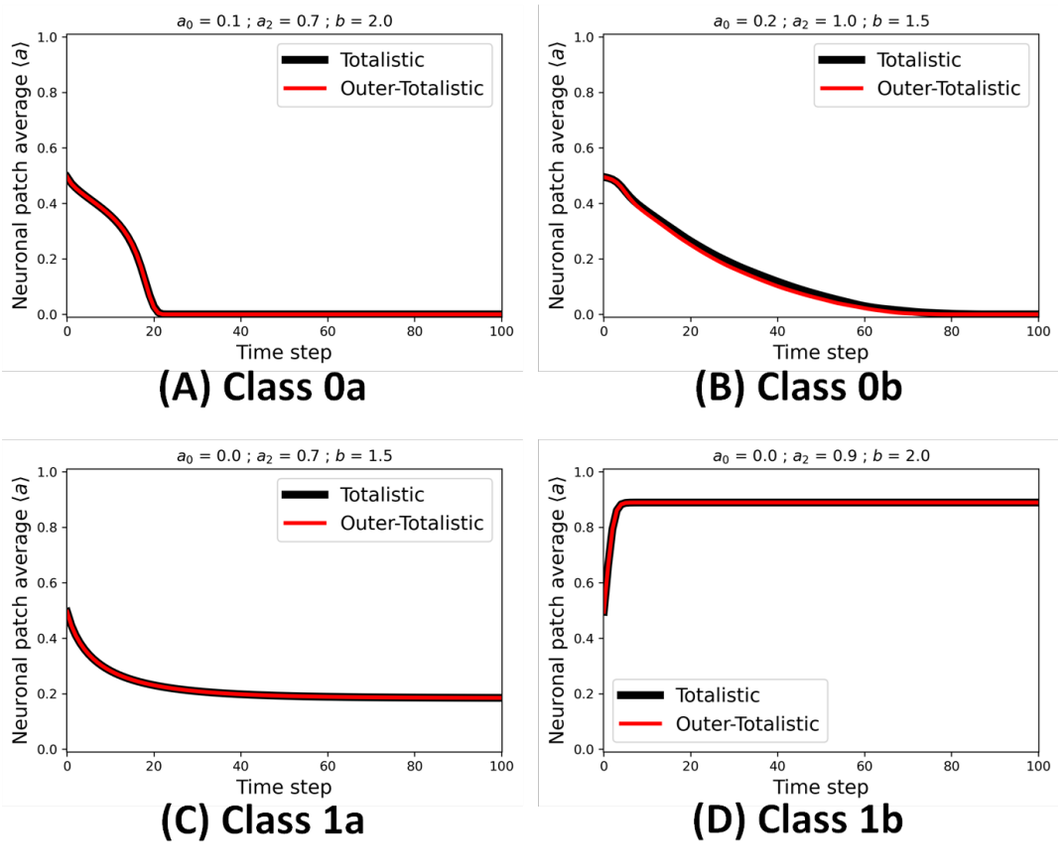


Figure 4.2: *Dynamics of a totalistic vs outer-totalistic with toroidal boundary conditions.* The classification of steady-state behavior remains unchanged under this neighborhood transformation.

Next, we fixed the setting to a totalistic neighborhood and compare between toroidal and a spherical lattice in Figure 4.3. No significant change in the dynamical classification is observed. Thus, we can fix totalistic toroidal lattice for the other analyses without the loss of generality.

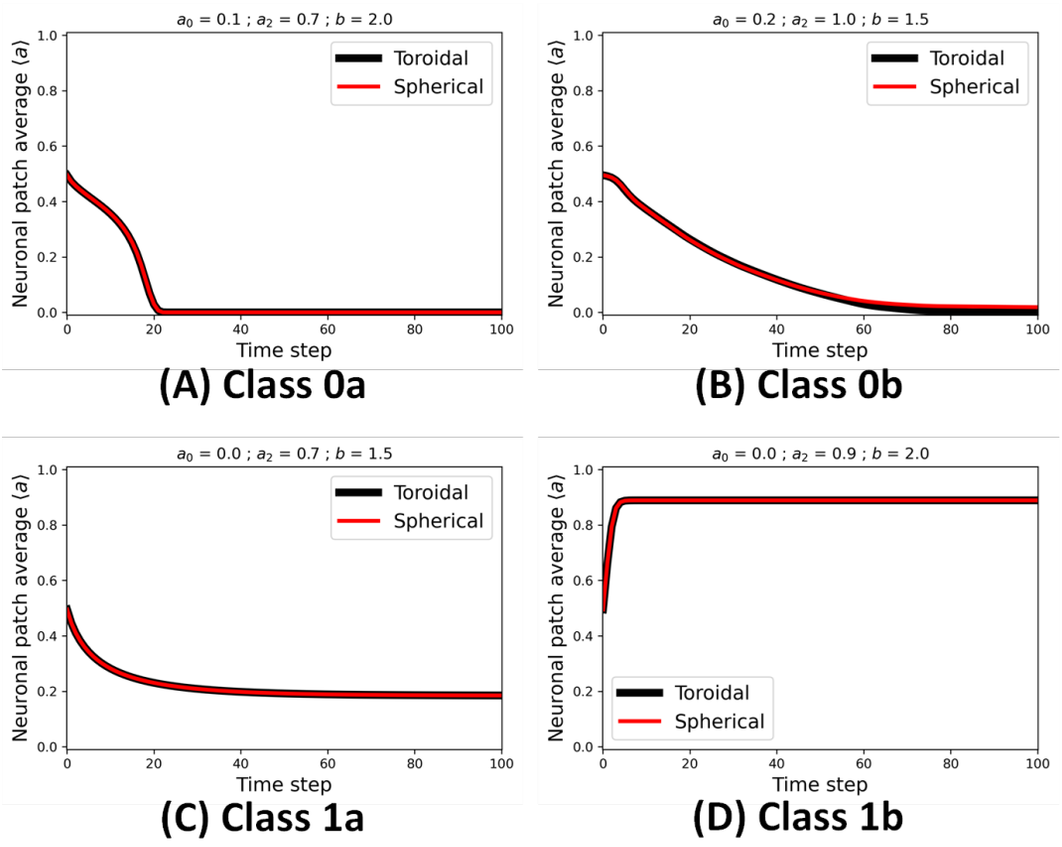


Figure 4.3: *Dynamics of a toroidal vs spherical with totalistic neighborhood conditions.* The classification of steady-state behavior remains unchanged under this lattice boundary transformation.

Extended Neighborhood and Boundary Conditions

Recall two other boundary conditions discussed in Section 2.4. Here, we investigated the dynamics of simulating multilayered NCA, and the effects of injecting external input to a totalistic toroidal NCA. For the multi-layered NCA, we compared the dynamics of a two-, three-, and four-layered network in Figure 4.4. No change in the steady-state dynamics of all CA subclasses except for Class 0b. For Class 0b, the higher the number of layers, the slower the decay of the activity. However, Class 0b still stabilizes at the zero steady-state given enough time steps.

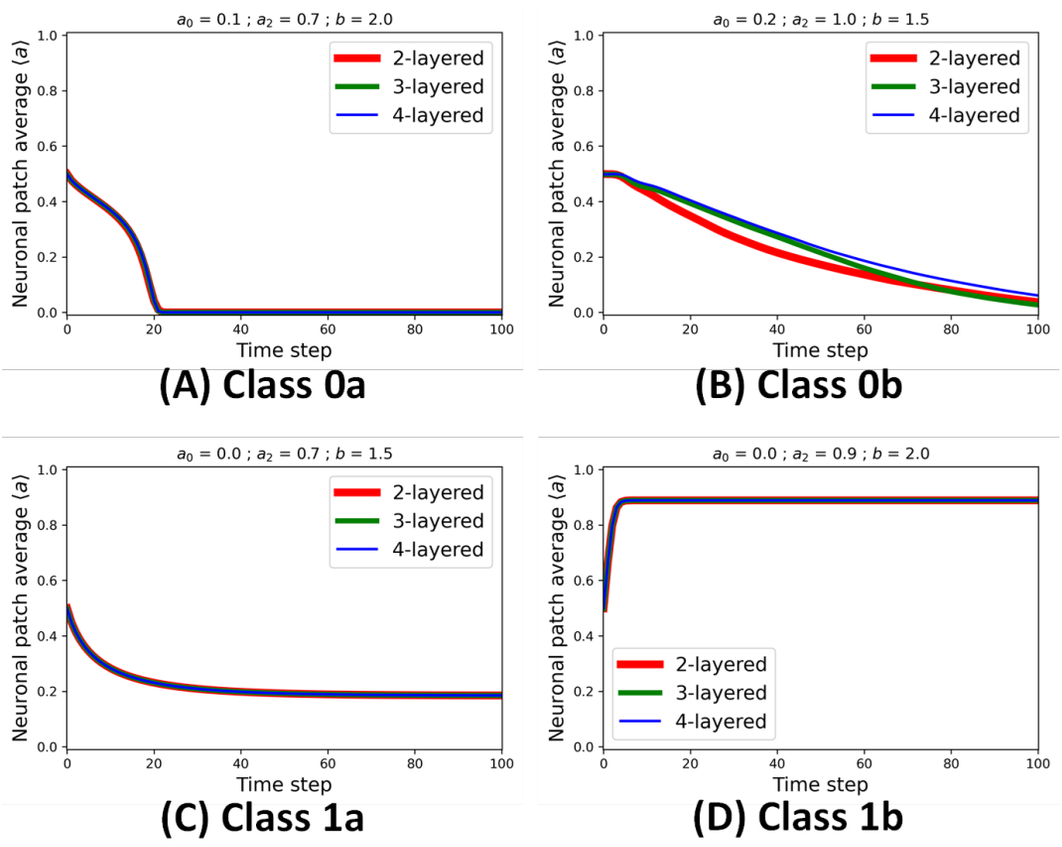


Figure 4.4: *Dynamics of a multilayered nonlinear NCA.* The classification of steady-state behavior remains unchanged.

In Figure 4.5, we investigated the effects of injecting sustained external input $a_{\text{ext}} = 1.0$ to $\xi \in \{1\%,\ 5\%,\ 10\%\}$ of the population of neurons N in the CA. Only Class 1b retained its steady-state behavior independent of ξ. For Class 1a, the sustained activation increases with ξ, making this class transition to Class 1b at high values of ξ.

Interestingly, both subclasses of Class 0 transitions to Class 1 when at least $\xi = 5\%$ of the neurons are injected with sustained input current. This means only a small number of neurons are needed to be probed to modify the internal dynamics of the nonlinear NCA. The steady-state activation of those in Class 0a increases with ξ. This steady-state behavior is analogous to Class 2 HH neurons, those that exhibit phasic-spiking.

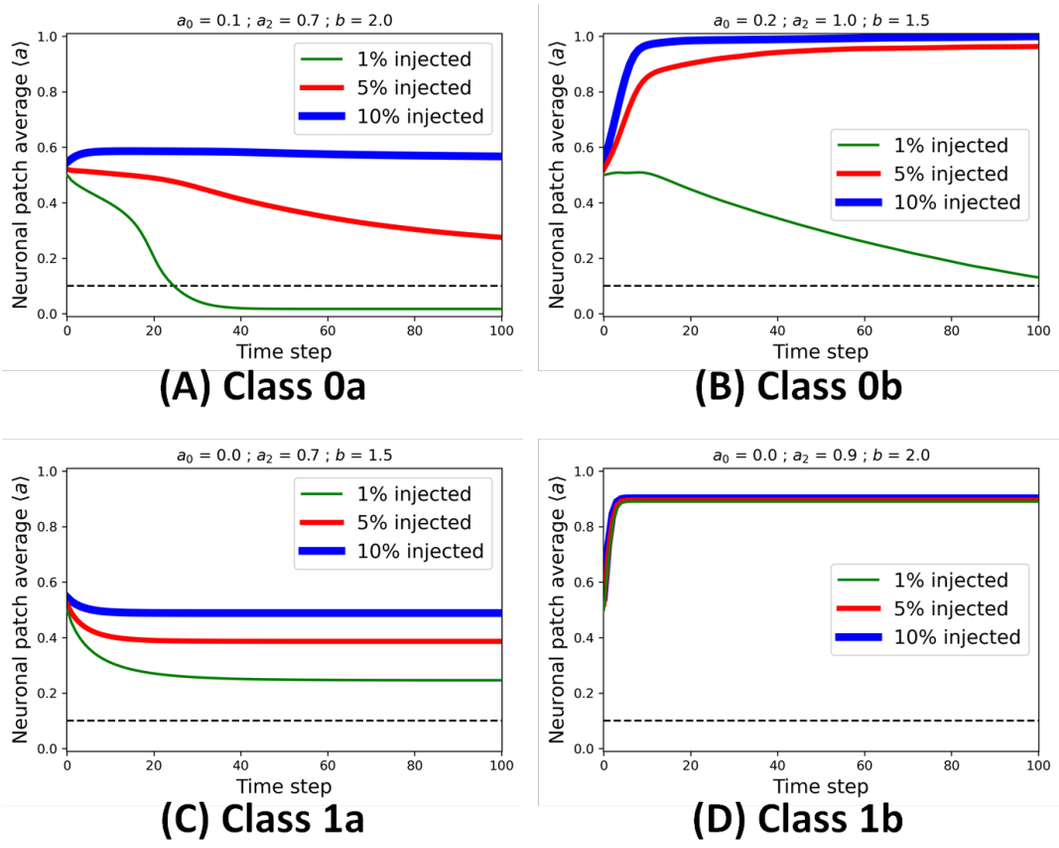


Figure 4.5: *Dynamics of a nonlinear NCA with external input.* The black dashed line indicates the activity of all the $\xi = 10\%$ neurons for reference. Both subclasses of Class 0 transitions to Class 1 when at least $\xi = 5\%$ of the neurons are injected with sustained input current.

4.3 Stability Analyses

In this section, we analyze only the neuronal CA with toroidal lattice and totalistic Moore neighborhood conditions. The analyses are composed of 1) local analysis, and 2) global analysis. In the local analysis, we investigate the interactions of a single neuron within its immediate neighbors in the CA. This includes obtaining the cobweb diagrams and the fixed points of the return map or the activation function. In the global analysis, we analyze the CA in its steady-state, obtaining the average global behavior of the neuronal network. This includes the phase space and bifurcation diagrams.

4.3.1 Local Analysis

First, we obtain the cobweb diagrams for each class at different initial states and trace the trajectories as shown in Figure 4.6.

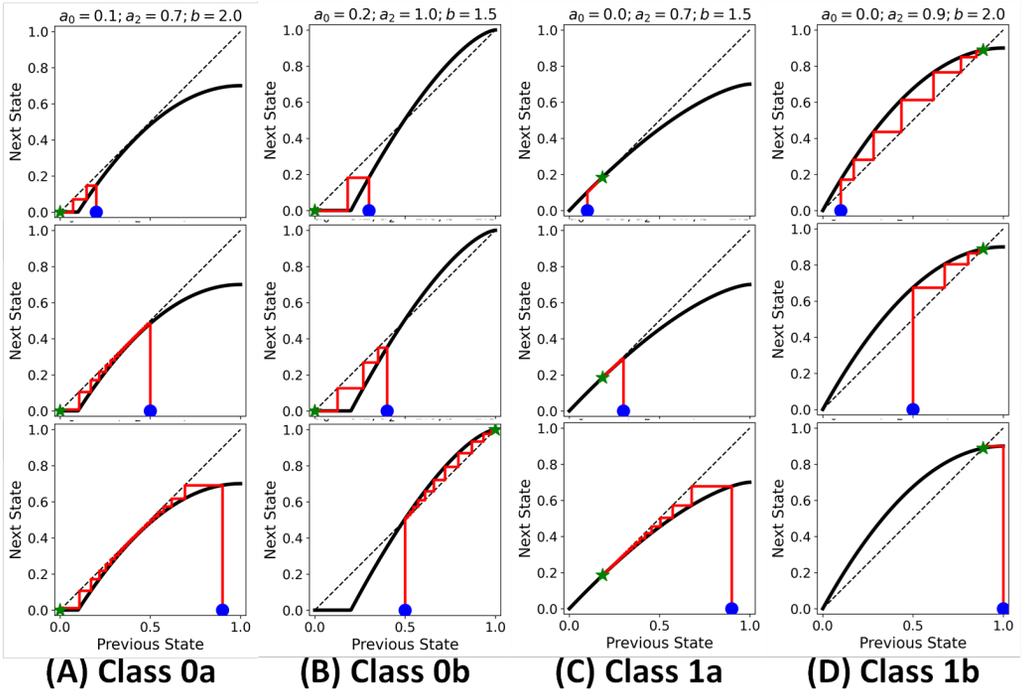


Figure 4.6: *Cobweb diagrams for nonlinear NCA.* Blue dot corresponds to the initial state while the green star indicates the steady-state or the final state. The red line traces the trajectory from the initial state to the final state.

In Class 0a, the activation function is completely below the diagonal, rendering any initial state to stabilize at zero. Hence a collection of these neurons will typically result to a fast-decay of activity towards the zero steady-state. In Class 0b, trajectories end up to either at zero or at one. On one hand, initial states below the intersection point end up at zero. On the other hand, initial state above the intersection point end up at one. So, on average, a collection of these neurons will either end up a quiescent or spiking steady-state. But those neurons initialized below the intersection point has a shorter

trajectory towards zero that these neurons dominate the behavior of the network, leading to a slower decay of activity towards the zero steady-state.

For Class 1a, any initial state will stabilize at the intersection point located at $a \approx 0.2$. This results to a low activation spiking steady-state. In Class 1b, the intersection point is located high enough at $a \approx 0.9$. Any initial state converges to this intersection point resulting to a high activation spiking steady-state.

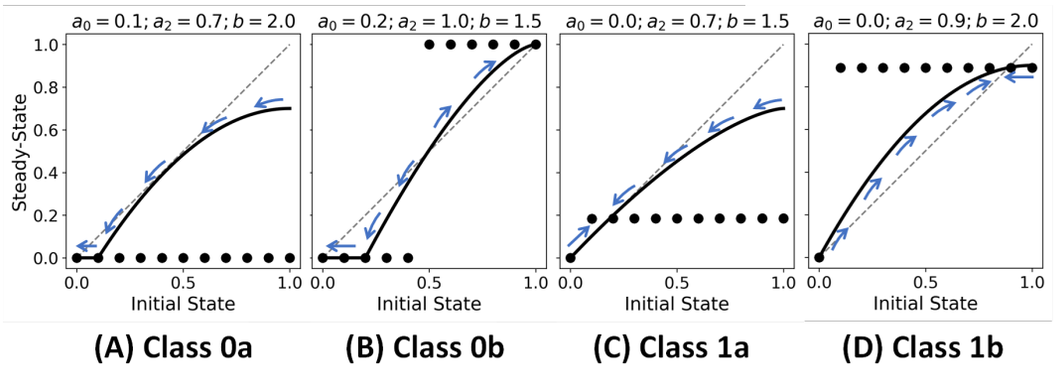


Figure 4.7: *Fixed points of nonlinear NCA.* The blue line indicates the transition of the system away from unstable fixed points towards stable fixed points.

The above behaviors is clearer if we plot the initial state vs the final state for each class as shown in Figure 4.7. The fixed points occur at the intersection points. and their stability are as follows.

0a: one stable fixed point at $a = 0$.

0b: two stable fixed points at $a \in [0, 1]$; one unstable fixed point at $a \approx 0.5$.

1a: one stable fixed point at $a \approx 0.2$; one unstable fixed point at $a = 0$.

1b: one stable fixed point at $a \approx 0.9$; one unstable fixed point at $a = 0$.

4.3.2 Global Analysis

Now we look at the neuronal CA as a whole and investigate the average behavior of the network at its steady-state. Since we have three parameters a_0, a_2, b, we construct three ways to two-dimensional parameter space, namely a_2-a_0, b-a_0, and b-a_2 parameter spaces, shown in Figure 4.8.

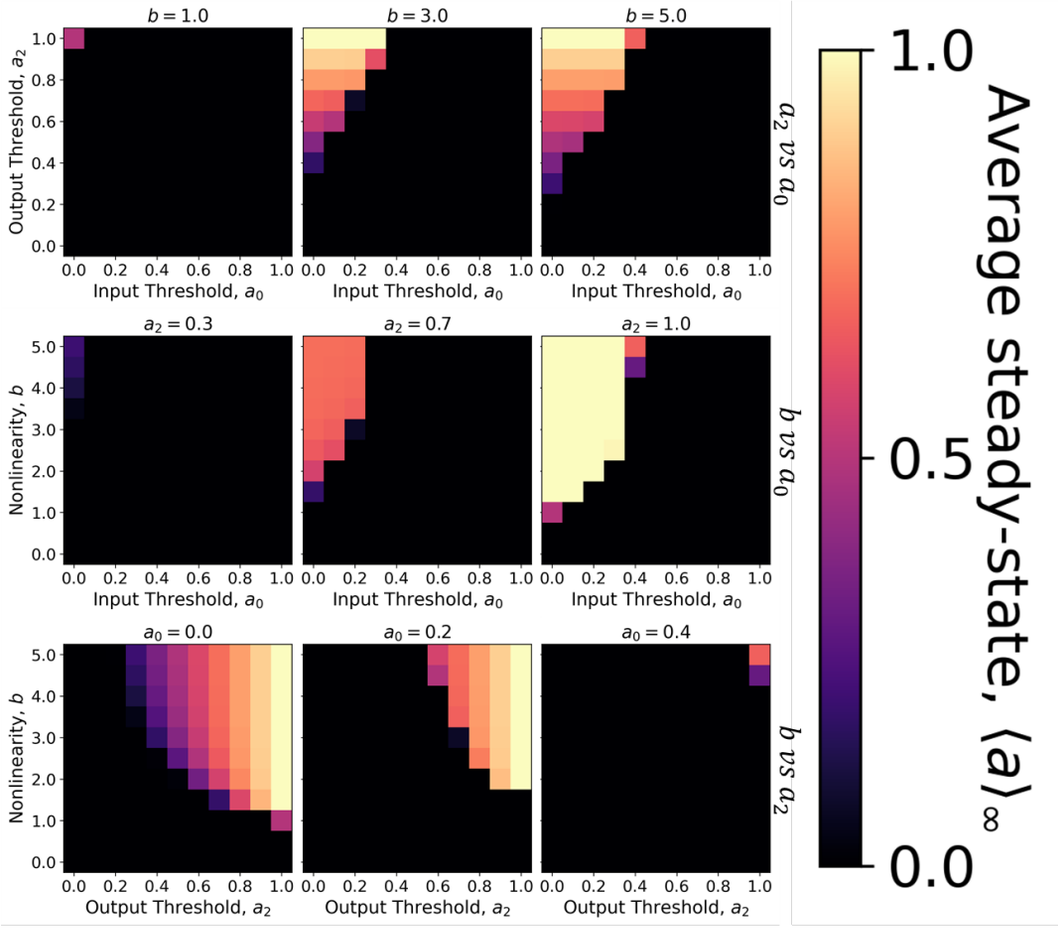


Figure 4.8: *Phase space diagrams for nonlinear NCA.* The average values $\langle a \rangle_\infty$ are taken from the last $t = 10$ time steps.

Black pixels in the phase space indicate Class 0 (zero steady-state) while colored pixels in indicate Class 1 (sustained spiking steady-state). In the a_2-a_0 parameter space, the phase boundary between Class 0 and Class 1 increases with the nonlinearity b. In the b-a_0 parameter space, the phase

boundary increases with a_2. However, in the b-a_2 parameter space, the phase boundary decreases with a_0. Altogether, we observed that NCA are classified as Class 1 at high values of b and a_2 but low values of a_0. This means that an activation function that spans most of the state space and is highly nonlinear would most likely behave with sustained spiking activity on its steady-state.

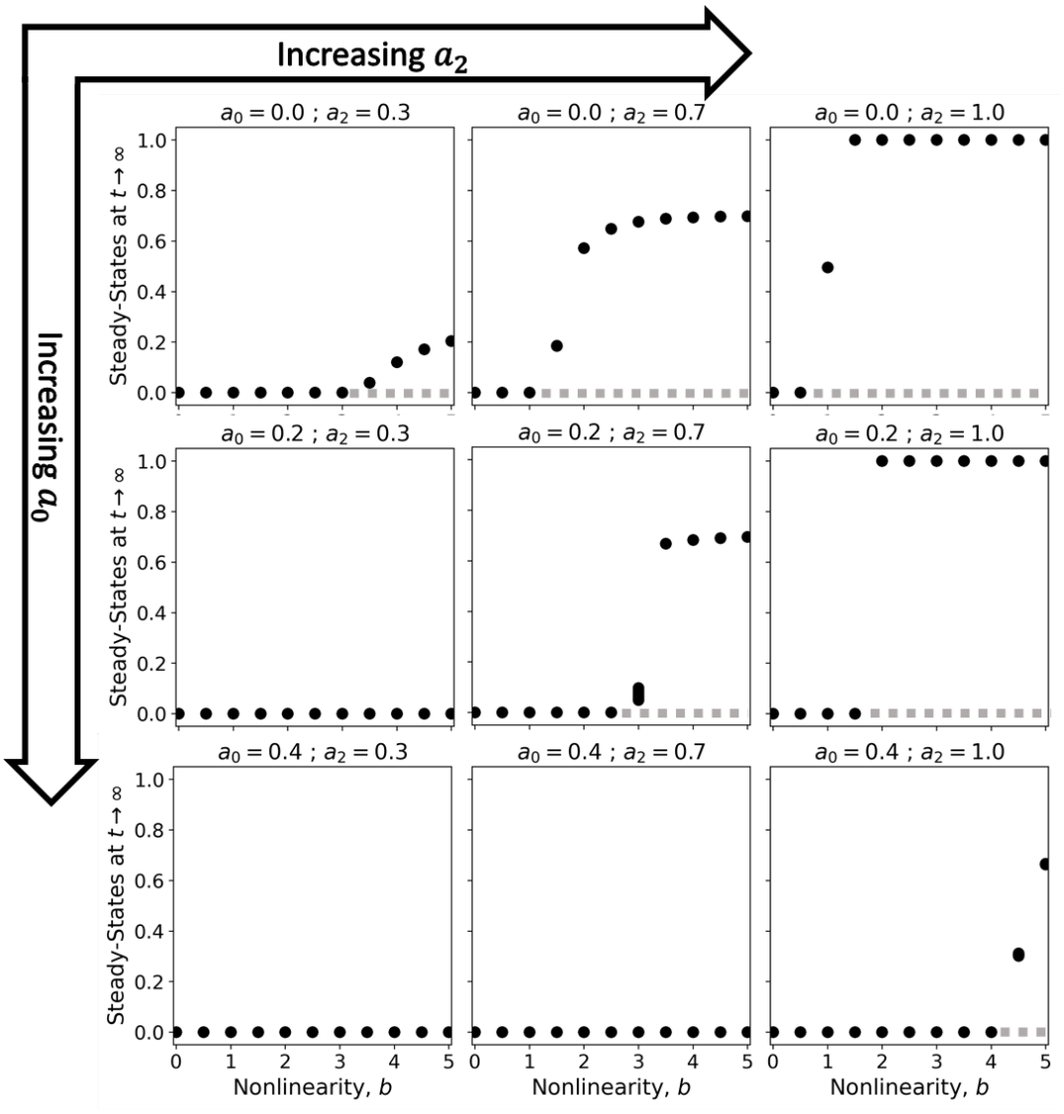


Figure 4.9: *Bifurcation diagrams of nonlinear NCA.* The steady-state values $\langle a \rangle_\infty$ are taken from the last $t = 10$ time steps. The gray dashed line corresponds to the branch of unstable fixed points.

Figure 4.9 shows the bifurcation diagrams with the nonlinearity b as the bifurcation parameter. Phase transition from Class 0 to Class 1 are evident when the bifurcation line starts to move up the diagram. The gray dashed line indicates the branch of unstable of fixed points. Although no forking or branching was observed, we expect a branch of fixed points below $a = 0$ but these states are inaccessible from our CA model.

The bifurcation diagrams also show that the average steady-state increases with the nonlinearity b. This is because as the nonlinearity increases, the activation function becomes more plateaued on a_2 resulting to a faster transition from low output activity to a higher output activity.

Chapter 5

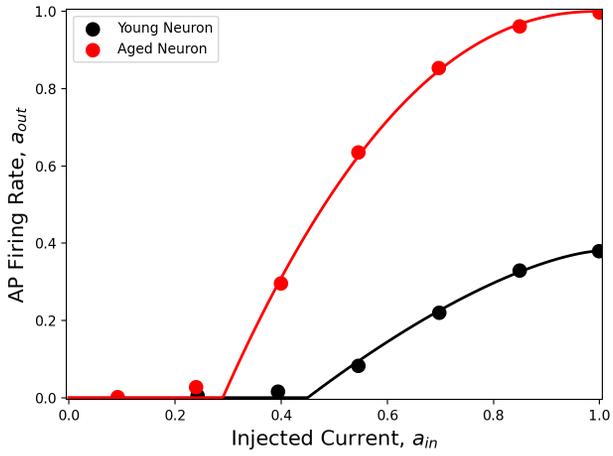
Nonlinear CA with Young and Aged Neurons

As an application, the nonlinear CA is modeled to incorporate properties that approximate young and aged neurons. This section follows the outline of the study by Coskren et al. [53] resulting to the firing rate of young and aged neurons for varying input current.

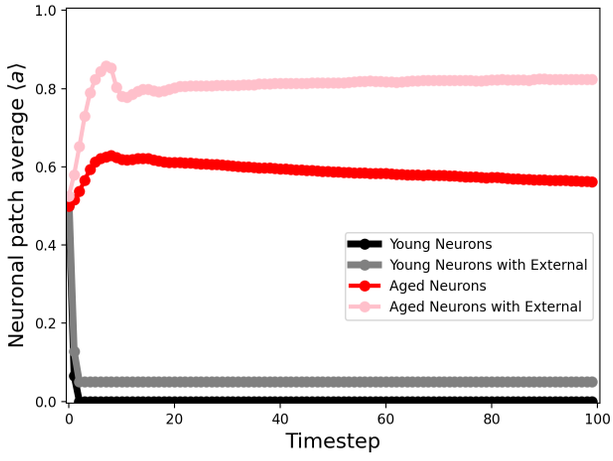
5.1 Activation Function and Dynamics

In their study, Coskren et al. obtained coronal slices of lateral prefrontal cortex (LPFC) collected from four young (8.0 ± 2.8 years old) and two aged (22.4 ± 3.4 years old) rhesus monkeys (*Macaca mulatta*). Resting membrane potential was determined by clamping the electrodes on the coronal slices when no current input is present. When current input of is injected, they recorded the membrane potential as a function of increasing current intensity. The recording $\Phi(t)$ was modeled via the HH model to obtain the corresponding firing rate. The dataset was fitted to the nonlinear activation function given by Equation 1.4 as shown in Figure 5.1(a).

Using the CA specifications (see Section 4.1) for the nonlinear NCA, we obtained the steady-state dynamics and plotted in Figure 5.1(b). In general, the aged neuronal network shows a spiking steady-state as opposed to the



(a) activation function



(b) steady-state dynamics

Figure 5.1: *Activation function and steady-state dynamics of young vs aged neurons.* Dataset for the firing rate is obtained from Coskren et al. (2015) [53].

quiescent steady-state behavior of younger neurons. When both systems are injected with sustained external input $a_{ext} = 1.0$ to $\xi = 5\%$ of the population, the aged neuronal network is greatly amplified revealing a much higher steady-state activity [54, 55]. Hence, aged neuronal systems does not require high intensities of stimuli for the activity to be amplified, unlike the younger systems. This conforms with the obtained firing rate of aged neurons,

which is generally higher than that of the younger ones [56, 57]. The presence of spiking states for aged at lower input currents are also observed by Coskren et al. [53].

5.2 Discrete Neuronal Response

To obtain the actual neuronal response, we distinguish between the continuous state s_a and the discrete state s_D of the neuron and apply the rule as follows.

$$\mathcal{R} = \{g(a_{i,j}) = \delta_{Q,s_D}\delta_{\nu,\mu_{QF}}F + \delta_{F,s_D}\mu_{FR}R + \delta_{R,s_D}\lfloor\nu + \mu_{QF}\rfloor Q\} \tag{5.1}$$

In this transformation, we define μ as the probability of the neuron to transition to the next spiking stage. Hence, we treat the a_t-values as the neuron's probability to fire at time step t. The transition rules are summarized in Figure 5.2 showing the conversion from a continuous to a discrete NCA.

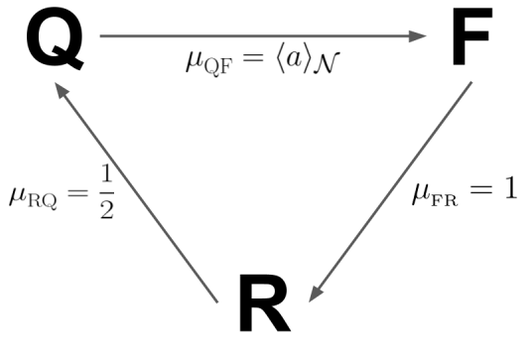
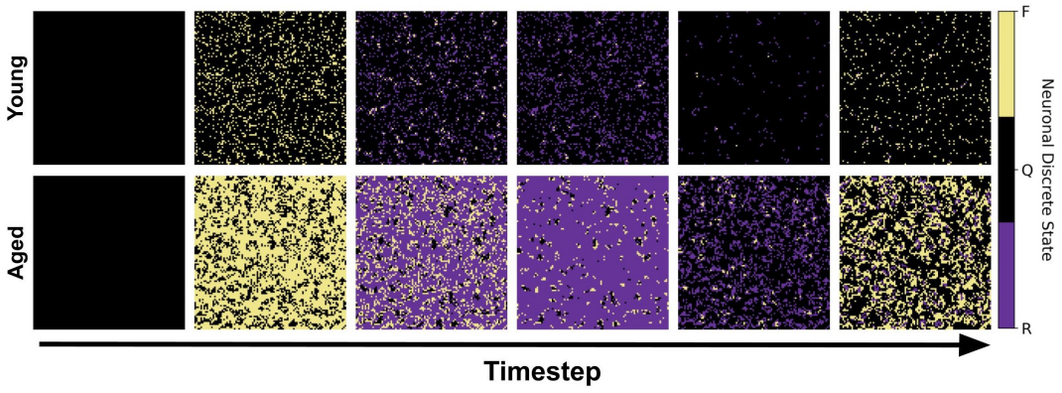
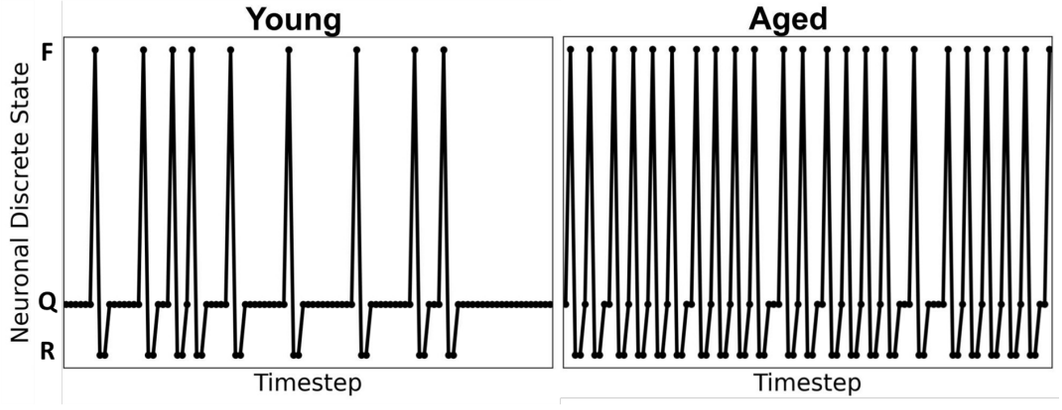


Figure 5.2: *State transformation rules from continuous to discrete NCA.* We treat the a_t-state as the probability of the neuron to fire at time step t.

The resulting spatiotemporal patterns for the young and aged neuronal networks are shown in Figure 5.3(a). Neurons with high spiking activity (indicated by 'F' states) are prominent to aged neuronal systems unlike in the younger neuronal networks.



(a) spatiotemporal evolution



(b) discrete response

Figure 5.3: *Spatiotemporal evolution and discrete neuronal response of young and aged neurons.* Activity of younger neurons is sustained if there is an constant external source of input. Older neurons experience a much higher sustained activity when the same amount of constant external input is injected.

Finally, we obtain a sample discrete response of a single neuron from each system as shown in Figure 5.3(b). Evidently, an aged neuron produces higher firing rate than that of the younger neuron, which again conforms with the observation by Coskren et al. [53]. The results discussed here are summarized in our previous work [29].

Chapter 6

Summary and Conclusions

Since the discovery of the neuron, there are still a lot of questions circling around the nervous system. How does the neuron interpret the signal? How does the neuron store information? Can we revive damaged neurons? Can we reverse the age of neurons? Do humans really need the 86 billion neurons or is it redundant to have as many? These and other questions remain unanswered even with the power of technology mankind has.

The ODE system developed by Hodgkin and Huxley as detailed in Chapter 1 best describes the spiking activity of a single neuron. The HH model was able to classify neural excitation as follows.

Class 1 Tonic-spiking. The neuron exhibits sustained spiking activity regardless of the strength of the stimulating current. The frequency of spikes is proportional to the strength of stimulating current, hence providing a continuous activation function.

Class 2 Phasic-spiking. The neuron undergoes two phase transitions as the stimulating current is increased. At very low currents, the neuron produces only one spike. At a critical current value, the neuron will spike only in the first few milliseconds of the stimulation period.

Class 3 Single-spiking. The neuron only has one phase transition: it produces only a single spike once the stimulating current is high enough.

But as we go up the each level of complexity within the nervous system, many new functions and behaviors emerge—behaviors that cannot be simply observed from the HH model. To replicate these behaviors, a network of HH neurons may be constructed but is severely restricted by the computational resource available.

To deal with larger neural systems, we outlined in Chapter 2 a neuronal cellular automata (NCA) model. One such model, the Brian's brain CA, produced patterns analogous to the transmission of signal within neuronal networks. The states in the Brian's brain CA are discretized according to the stages of neuronal spiking. This thesis presented an improved version of this NCA, incorporating a continuous state that corresponds to the net activity of each neuron in the network. Our NCA model utilized the HH activation function as its update rule to simulate the behavior of individual cells within the network. This state may be interpreted as the input current, the probability to fire, the firing rate, or the voltage output of the neuron.

We also showed the efficiency, both time and memory, of using cellular automata model versus solving system of ODEs with various algorithms. The forward Euler method and its derivative algorithms showed quadratic time complexity $T \sim \mathcal{O}(N^2)$ while the NCA model is linear in time $T \sim \mathcal{O}(N^1)$. In terms of memory, while the Blue Gene/P supercomputer used $147,456$ CPUs and 144 TB of main memory to simulate 10^9 neurons of a cat, our NCA model is powered only by 1 CPU (with 8-cores) and 16 GB of memory (RAM) which can easily handle a lattice with $N \sim 10^6$. For comparison, the supercomputer comes with an efficiency of roughly $\sim 6 \times 10^3$ neurons/CPU and ~ 144 KB/neuron, while our NCA model yields an efficiency of roughly 4×10^6 neurons/CPU and ~ 4 bytes/neuron) and with memory to spare for other calculations!

With the CA model, we explored various lattice boundary and neighborhood conditions from 2D to quasi-3D configuration. In a previous work, we proposed an NCA model with a linear activation function and observed three

classes of neuronal patch steady-state behavior.

1. Class 0: Quiescent Steady-State:
   (a) Fast-decay ; (b) Slow-decay
2. Class 1: Spiking Steady-State:
   (a) With random patterns ; (b) With exploding patterns
3. Class 2: Oscillating Steady-State

The patterns in Class 0 and Class 1 visually replicates the transmission signal in a continuous space while the periodic steady-state, which is only accessible when the activation function is negatively-sloped, seemingly captures the behavior of epileptic neurons.

In Chapter 3, we take the logistic equation as an example to introduce essential concepts and analyses within the bifurcation theory. The theory is just one way to analyze dynamics of systems with complex behavior and emergence. The analyses can be grouped into two: 1) local analysis, and 2) global analysis. The local analysis reveals short-range interactions and the global analysis shows the long-range interactions.

In the local analysis, the interactions of a single neuron within its immediate neighbors (locality) are investigated. This can be shown by the trajectory over time of the cell's state in the cobweb diagrams. On the cobweb diagram, the intersection points for which the mapping function intersects with the diagonal are the fixed points of the system. By determining the behavior of the trajectory around the fixed points, we can determine the stability at that fixed point. A trajectory that moves towards (away) from the fixed point is said to be stable (unstable). Moreover, if the trajectory oscillates around a fixed point, then we have a limit cycle.

In the global analysis, we analyze the CA in its steady-state, obtaining the average global behavior of the neuronal network. This includes the phase space and bifurcation diagrams. The phase space displays the phases a system may exhibit, which can be distinguished by the phase boundary. When crossing this boundary, the system will undergo phase transition that will

change the steady-state behavior. However, some information may not be visible in the phase space such as periodic behavior. The periodic and chaotic behavior are best revealed by the bifurcation diagrams. Different types of bifurcation describe the phase transitions of the system when moving around the fixed points.

Chapter 4 combines all the concepts from neuroscience theory, cellular automata theory, and bifurcation theory, into one powerful model. Here, we outlined a neuronal CA model that utilizes the nonlinear approximation to the HH activation function as the update rule. We simulated the CA in various lattice boundary and neighborhood conditions revealing the following classes of steady-state behavior.

1. Class 0: Quiescent Steady-State:
   (a) Fast-decay ; (b) Slow-decay
2. Class 1: Spiking Steady-State:
   (a) Low activation ; (b) High activation

The steady-state behavior remains unchanged except only when an external input is introduced in the system. We found that when at least 5% of the network is injected with constant input current, Class 0 transitions to Class 1 drastically. This behavior is also observed from the experiments on the young and aged neuronal networks by Coskren et al as discussed in Chapter 5. Aged neuronal systems does not require high intensities of stimuli for the activity to be amplified, unlike the younger systems. Hence, aged neurons, in general have higher firing rate than that of younger neurons.

We also discussed in Chapter 4 the cobweb and bifurcation analysis to the nonlinear NCA, we found the following fixed points and their stability.

0a: one stable fixed point at $a = 0$.

0b: two stable fixed points at $a \in [0, 1]$; one unstable fixed point at the second intersection point.

1a: one stable fixed point at the second intersection point ; one unstable fixed point at $a = 0$.

1b: one stable fixed point at the second intersection point ; one unstable fixed point at $a = 0$.

The location of the stable fixed points of the activation function easily determines the neuron's steady-state behavior. Since no limit cycles was found, no bifurcation is observed. This is backed by the bifurcation diagrams, which shows only one branch of stable of fixed points and one branch of unstable fixed points.

Meanwhile, the phase space diagrams reveal how the system transitions between the classes when we move along the parameter space. The phase boundary increases with the parameters b and a_2, but decreases with a_0. This means that Class 1 CA favors a highly nonlinear activation function that is more spread above the diagonal.

Chapter 5 demonstrates the power of the NCA model in simulating spiking activity of different kinds of neurons. Given any type of neuron, if the firing rate as a function of input current is known, then we can easily modify the rules of the NCA model to investigate the dynamics of these neurons in different neighborhood and boundary conditions. This means that we can visualize and compare the dynamics for various kinds of 2D and even quasi-3D topologies.

In conclusion, what we have obtained in this work is a comprehensive study of the neuronal CA that utilizes the nonlinear approximation to the HH activation function as its update rule. We explored different configurations of the lattice and the neighborhood extending up to quasi-3D networks. The dynamics of the steady-state behavior of the CA was analyzed using various methods developed under bifurcation theory. Overall, this work aims to provide insight and computational groundwork of a simplified large-scale neuronal network for future research and development.

Appendix A: Published Articles

The following articles are presented in order:

Jan 2022 Young and Aged Neuronal Tissue Dynamics With a Simplified Neuronal Patch Cellular Automata Model. **Journal**: Frontiers in Neuroinformatics (5-year Impact Factor: 4.602)

Oct 2021 Simplified cellular automata model of neuronal patch dynamics with generalized non-linear cell response. **Proceedings**: 39th Samahang Pisika ng Pilipinas

Oct 2020 Verhulst and bifurcation analyses of a neuronal network on an outer-totalistic toroidal cellular automata. **Proceedings**: 38th Samahang Pisika ng Pilipinas

May 2019 Totalistic cellular automata model of a neuronal network on a spherical surface. **Proceedings**: 37th Samahang Pisika ng Pilipinas

June 2018 Proposed cellular automaton model for a neuronal patch with a thresholded linear activation function. **Proceedings**: 36th Samahang Pisika ng Pilipinas

The following posters are presented in order:

Oct 2019 An outer-totalistic 2D and quasi-3D cellular automata simplistic models of neuronal patches. **Conference**: Brain Connects 2019 and the 9th Neuroscience International Symposium

Oct 2019 Classification of the dynamics of an outer-totalistic 2D and quasi-3D cellular automata simplistic models of neuronal patches. **Conference**: 16th International Conference on Molecular Systems Biology

frontiers in Neuroinformatics

ORIGINAL RESEARCH
published: 07 January 2022
doi: 10.3389/fninf.2021.763560

Young and Aged Neuronal Tissue Dynamics With a Simplified Neuronal Patch Cellular Automata Model

Reinier Xander A. Ramos[1], *Jacqueline C. Dominguez*[2,3] *and Johnrob Y. Bantang*[1,4*]

[1] *Instrumentation Physics Laboratory, National Institute of Physics, College of Science, University of the Philippines, Quezon City, Philippines,* [2] *Institute for Neurosciences, St Luke's Medical Center, Quezon City, Philippines,* [3] *Elderly and Dementia Care, Institute for Dementia Care Asia, Quezon City, Philippines,* [4] *Computational Science Research Center, University of the Philippines, Quezon City, Philippines*

Realistic single-cell neuronal dynamics are typically obtained by solving models that involve solving a set of differential equations similar to the Hodgkin-Huxley (HH) system. However, realistic simulations of neuronal tissue dynamics —especially at the organ level, the brain— can become intractable due to an explosion in the number of equations to be solved simultaneously. Consequently, such efforts of modeling tissue- or organ-level systems require a lot of computational time and the need for large computational resources. Here, we propose to utilize a cellular automata (CA) model as an efficient way of modeling a large number of neurons reducing both the computational time and memory requirement. First, a first-order approximation of the response function of each HH neuron is obtained and used as the response-curve automaton rule. We then considered a system where an external input is in a few cells. We utilize a Moore neighborhood (both totalistic and outer-totalistic rules) for the CA system used. The resulting steady-state dynamics of a two-dimensional (2D) neuronal patch of size $1,024 \times 1,024$ cells can be classified into three classes: (1) Class 0–inactive, (2) Class 1–spiking, and (3) Class 2–oscillatory. We also present results for different quasi-3D configurations starting from the 2D lattice and show that this classification is robust. The numerical modeling approach can find applications in the analysis of neuronal dynamics in mesoscopic scales in the brain (patch or regional). The method is applied to compare the dynamical properties of the young and aged population of neurons. The resulting dynamics of the aged population shows higher average steady-state activity $\langle a(t \to \infty)\rangle$ than the younger population. The average steady-state activity $\langle a(t \to \infty)\rangle$ is significantly simplified when the aged population is subjected to external input. The result conforms to the empirical data with aged neurons exhibiting higher firing rates as well as the presence of firing activity for aged neurons stimulated with lower external current.

Keywords: neuronal dynamics, continuous cellular automata, brain, numerical model, activation function, aged neurons

OPEN ACCESS

Edited by:
Toshiharu Nakai,
Osaka University, Japan

Reviewed by:
Andrei Dragomir,
National University of Singapore, Singapore
Ergin Yilmaz,
Bulent Ecevit University, Turkey

***Correspondence:**
Johnrob Y. Bantang
jybantang@up.edu.ph

Received: *25 August 2021*
Accepted: *06 December 2021*
Published: *07 January 2022*

Citation:
Ramos RXA, Dominguez JC and Bantang JY (2022) Young and Aged Neuronal Tissue Dynamics With a Simplified Neuronal Patch Cellular Automata Model. Front. Neuroinform. 15:763560. doi: 10.3389/fninf.2021.763560

1. INTRODUCTION

Since the development of the first neuronal model by Louis Lapicque in 1907, most neuronal models we have today use a set of ordinary differential equations (ODEs) to model the dynamics of neurons (Lapicque, 1907; Brunel and Van Rossum, 2007). The Nobel-prize winning Hodgkin-Huxley (HH) model describes the relationship between the membrane potential of the neuron and

the flow of ions across the membrane normally via the ion channels (Hodgkin and Huxley, 1952; Gerstner et al., 2014). The HH model is successfully used to describe the dynamics of a squid giant axon and even the Purkinje fibers in the heart (Noble, 1962). Other models such as Dalton and FitzHugh (1960), Nagumo et al. (1962) and Morris and Lecar (1981) models were improvisations and simplifications on the HH model. While these models are good representations of a neuronal response, it is a challenge for us to construct a simple model useful in describing the behavior of a large neuronal population. HH neurons can be arbitrarily interconnected (Pang and Bantang, 2015) but simulations for large numbers of neurons take long computational run time and need high computing resources since they require solving many coupled ODEs and saving numerous system variables.

One study involves cortical simulations of 10^9 neurons of a cat using Blue Gene/P supercomputer (Ananthanarayanan et al., 2009). The simulations were powered by 147, 456 CPUs and 144 TB of main memory (roughly $\sim 6 \times 10^3$ neurons/CPU, $\sim$144 KB/neuron). In this study, we propose simple cellular automata models to simulate many interconnected neurons that will help investigate integrated dynamics of up to millions (10^6) of neurons using lower CPU and GPU requirements. Our simulations are powered with 1 CPU and 16 GB of memory (RAM) (roughly $\sim 10^6$ neuron/CPU, $\sim$16 KB/neuron). The Blue Brain project primarily uses the NEURON simulation environment to accomplish their feat. NEURON mainly solves ODE-based models with data-driven parameters. However, solving ODEs differs from the cellular automata (CA)-based models. CA models can employ a look-up-table-based algorithm that is usually faster than solving ODEs.

Cellular automaton modeling paradigm was first developed in the late 1940's by Stanislaw Ulam and John von Neumann (von Neumann, 1966). It became popular after it was used to model Conway's Game of Life in the 1970's. A CA system $\mathcal{A}$ consists of the set $\mathcal{C}$ of agents or "cells" c ($c \in \mathcal{C}$) arranged in a lattice $\mathcal{L}$ with a specified neighborhood set $\mathcal{N} = \mathcal{C}^{n+1}$ where n is the number of neighbors of any given cell. Certain boundary conditions are also applied depending on the properties of the physical system being modeled (Wolfram, 2002; Arciaga et al., 2009). These cells have assigned state s, typically obtained from a finite state binary set such $\mathcal{S} = \{0, 1\}$, being the simplest. The "0" and "1" states usually represent either "dead" or "alive," or for our present case of neuronal dynamics represent "resting" or "spiking" (active), respectively. Each neighborhood has a unique state $\overline{s} \in \mathcal{S}^{n+1}$.

The various dynamics of a CA model also emerge from the rules applied to the lattice. In this work, we investigate a CA system with a first-order linear approximation to the HH neuronal response as our rule for each cell. The activation function is further discussed in section 2.1. We perform different analyses (spatiotemporal, cobweb, bifurcation) on the CA system to classify the observed dynamics. This lays the groundwork of our proposed model that can be extended to future directions. In section 7, we extended our model into a nonlinear activation function, which is used to better fit the response of young and aged neurons of a rhesus monkey (Coskren et al., 2015).

Aged neurons have distinctly less myelin and shorter axon internodal distance leading to reduced conduction velocity (Peters, 2007). Dysregulated signaling pathways in oligodendroglia and the loss of reregenerative capacity of oligodendrocyte progenitor cells are thought to be the major cause for myelin loss (Rivera et al., 2021). At the synapse, dendritic spines where majority of excitatory synaptic processes occur are smaller and lesser (Pannese, 2011) but are functionally intact to make synaptic connections. The connections may be weaker but exhibit lesser capacity for short-term plasticity (Mostany et al., 2013). Presumably, the shrinkage in the density of the dendritic spine impacts excitatory synaptic activity in neuronal circuits and accounts for the cognitive changes observed in older adults even in the absence of pathology. However, in the light of reports of increase in action potential firing rates (excitability) in aged neurons, there is need for studies to further understand the dynamics of cell-to-cell communication and open avenues for potential interventions to mitigate the effects of brain aging. In a study on rhesus monkey prefrontal cortex (Coskren et al., 2015), it was found that aged neurons typically have higher action potential (AP) firing rates compared to younger neurons. The empirical data from the study is used as an application of our CA model.

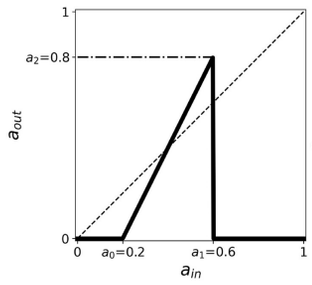
2. CONTINUOUS CELLULAR AUTOMATA MODEL OF A NEURONAL PATCH

As a CA model, neurons are arranged in a two-dimensional lattice $\mathcal{L}$ composed of $1,024 \times 1,024$ cells. This choice of lattice size is one of the highest possible in a common computing device (without the need of high-performance computing). The resulting dynamics does not change with varying lattice size (Ramos and Bantang, 2018, 2019c). However, the computing performance is compared in section 9. The state of s each neuron is represented by a real number a (stands for activity) which ranges from 0 to 1, thus forming a continuous-state CA. The state of each neuron is initialized by assigning a random value to the CA state drawn from a uniform distribution such that $a_{i,j} \in [0, 1]$ for all CA cells in the system ($i, j \in [1, 1024]$).

At each timestep, the average state of the neighborhood of a given cell is taken as the cell's input a_{in} or stimulus. The response of the current cell is obtained from the mapping of a_{in} into its corresponding output a_{out} or response. A generalized linear activation or response function is shown in **Figure 1**. The various modes of neighborhood and boundary conditions are discussed in section 2.2 and the activation function is discussed in section 2.1 below. We found that a value of 100 timesteps is enough to achieve steady-state for any initial state, and that randomizing the initial location of active cells does not affect the dynamical results of our model (Ramos and Bantang, 2018; Ramos, 2019).

2.1. Activation Function

The activation function used in the CA model is mainly derived from the response function of the HH model. Many other neuronal models such as leaky integrate-and-fire (Tal and Schwartz, 1997; Gerstner et al., 2014) and Wilson-Cowan (Wilson and Cowan, 1972) exhibit a similar trend of neuronal firing



$$a_{\text{out}} = \begin{cases} 0 & a_{\text{in}} < a_0 \\ \frac{a_2(a_{\text{in}}-a_0)}{a_1-a_0} & a_{\text{in}} \in [a_0, a_1] \\ 0 & a_{\text{in}} > a_1 \end{cases}, \quad (1)$$

FIGURE 1 | A simplified neuronal response used for the CA model of neurons. The output a_{out} is related to the probability of the neuron to fire given the input a_{in}. The plot shows such functions for threshold values $a_0 = 0.2$, $a_1 = 0.6$, $a_2 = 0.8$. The dashed line corresponds to $a_{\text{out}} = a_{\text{in}}$ useful in the cobweb analysis.

rate with increasing input current. Three main properties of the activation function can be observed:

1. Two thresholds (a minimum and a maximum) in the input are present indicating that neurons fire only when stimulated by an input current between these two thresholds. We, respectively, assign for these the thresholds a_0 and a_1, the minimum and maximum.
2. The firing rate monotonically increases whenever the input current is between a_0 and a_1; the firing rate is zero otherwise.
3. A maximum threshold in the output is present limiting the firing rate values for the entire range of a_{in}. We assign this as a_2.

The thresholds are incorporated into the activation function and are simplified by taking the first-order approximation as described in **Figure 1**. The parameter thresholds are varied from 0 to 1 with a step size of 0.1. The condition $a_0 = a_1$ results in a trivial mapping $a_{\text{out}} = 0$, for all a_{in}-values. The resulting equation for the neuronal activation function is given by:

$$a_{\text{out}} = \begin{cases} 0 & a_{\text{in}} < a_0 \\ \frac{a_2(a_{\text{in}}-a_0)}{a_1-a_0} & a_{\text{in}} \in [a_0, a_1] \\ 0 & a_{\text{in}} > a_1 \end{cases} \quad (1)$$

We performed an exhaustive search by varying each parameter in $\{a_0, a_1, a_2\}$ from 0 to 1 with increments of 0.1 (Ramos and Bantang, 2018, 2019c). The resulting steady-state dynamics for all possible combinations of $\{a_0, a_1, a_2\}$ in this scheme were classified into one of the types discussed in section 3.

2.2. Neighborhood and Boundary Conditions

Two often used neighborhood configurations in CA models are the von Neumann and the Moore neighborhoods (Wolfram, 2002). **Figure 2A**, on one hand, shows a von Neumann setting. In this case, the central cell of any 3 × 3 subset of the lattice is connected to the adjacent cells in the primary directions (4 neighbors: left, right, top, and bottom) with respect to the cell. **Figure 2B**, on the other hand, shows a Moore neighborhood setting. This time, the central cell is connected to the adjacent cells in the primary and secondary directions (including the diagonal directions, total of 8 neighbors). Moore neighborhood is used in the model since a biological neuron is typically connected to all neighboring cells in the 2D space (Hawick and Scogings, 2011). Two types of Moore neighborhood configurations are considered: totalistic and outer-totalistic. The only difference between these configurations is that the outer-totalistic setting has the central cell of the 3 × 3 subset included in the neighborhood state (see **Figure 2C**).

The boundary conditions describe how the cells at the edge of the lattice behave. Two types of boundary conditions were considered: toroidal and spherical boundaries. With the toroidal boundary condition, the cells on the leftmost column are connected to the rightmost column, and the top row is connected to the bottom row. This produces a wrap-around effect on our automaton as shown in **Figure 3A**. For the spherical setting (Ramos and Bantang, 2019c), the square lattice is projected on the surface of a sphere (Mercator projection) as shown in **Figure 3B**. Observe that the cells in the top (and bottom) row are fully connected to each other becoming a pole, while in the middle rows neighbors wrap around.

2.3. Two-Layered Lattice and the External Input

To analyze a quasi three-dimensional (3D) neighborhood, we extended our analysis to a two-layered automaton for both toroidal and spherical boundary conditions (Ramos and Bantang, 2019a,b). The intra-layer connection has a Moore neighborhood setting, while the inter-layer connection is a direct overlay between the layers. The neighborhood conditions for this two-layer lattice is visualized in **Figure 3C**. For systems with the number of layers greater than two, the topmost and bottommost layer are connected as if the bottommost layer is stacked above the topmost layer.

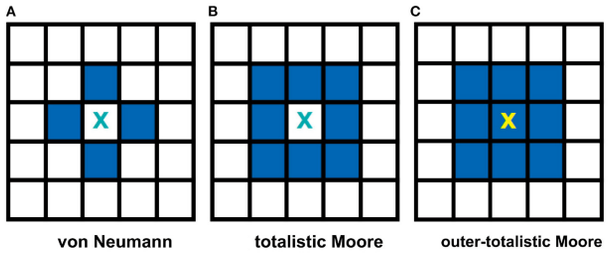


FIGURE 2 | Common neighborhood configurations used in CA theory. The shaded cells show the neighbors of cell X for each type of neighborhood. The von Neumann neighborhood **(A)** of cell X consists of the four cells in the primary directions, while Moore **(B)** extends it to the secondary directions. In an outer-totalistic setting **(C)**, the cell X itself is included in the neighborhood. In this work, Moore neighborhood is used because a biological neuron is typically connected to all neighboring cells in the 2D space.

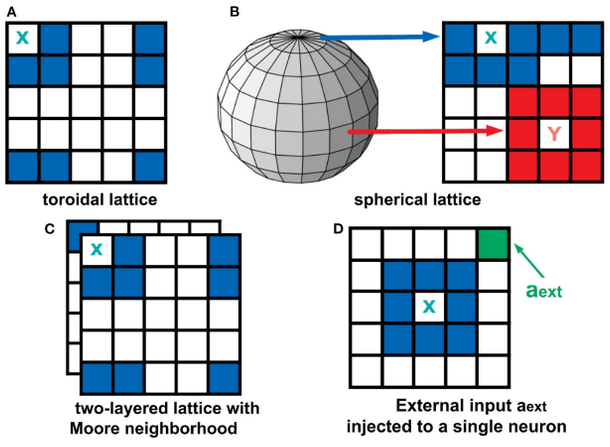


FIGURE 3 | Top row: Boundary conditions used in this study. In a toroidal lattice **(A)**, we wrap-around the top and bottom rows, and the leftmost and rightmost parts of the grid. For the spherical lattice **(B)**, a Mercator projection was used to draw the lattice on the sphere's surface. Bottom row: Extended neighborhood and boundary conditions explored in this work. For the two-layered lattice **(C)**, the intra-layer connection is Moore, but the inter-layer connection is the overlay between layers. For the analysis of external input **(D)**, a fraction of neurons in the population are set to be always active $a_{\text{ext}} = 1$.

A CA system (Ramos and Bantang, 2019a) with a constant external input $a_{\text{in}} = a_{\text{ext}} = 1$ injected to one of the neurons c_{ext}, shown in **Figure 3D**, is also analyzed. In this case, the neuron c_{ext} is always in spiking state since $a = 1$ at all times.

3. NUMERICAL EXPERIMENTS

We first examined the dynamics of the neuronal CA using Moore toroidal boundary condition. The average neuronal patch activity $\langle a \rangle$ is obtained for each timestep and plotted as shown in **Figure 4**. We observed two types of steady-state dynamics: a quiescent or zero steady-state; and a spiking or nonzero steady-state. These steady-state trends are also observed in the HH model (as well as Morris-Lecar) as Type I and Type II neurons, respectively (Hodgkin and Huxley, 1952; Morris and Lecar, 1981; Gerstner et al., 2014). The steady-state dynamics is the same for totalistic and outer-totalistic neighborhoods. Samples of spatiotemporal activity of the neuronal CA are shown in

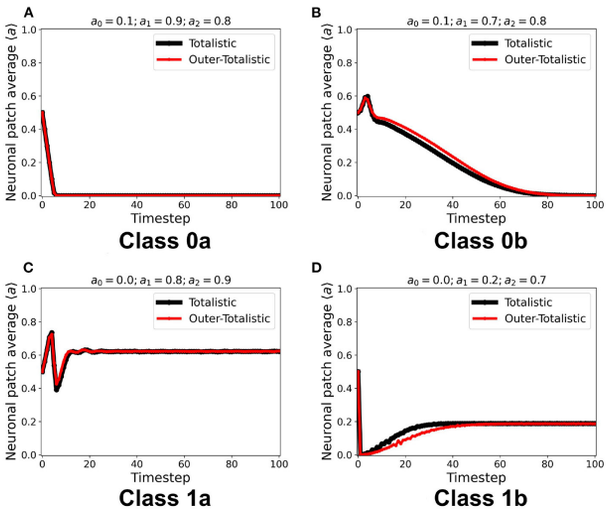


FIGURE 4 | (A–D) Representative steady-state dynamics for each neuronal CA class. The steady-state is taken as the average neuronal activity of the patch at that timestep. The black solid line shows the average neuronal activity using totalistic rules while the red solid line corresponds to the outer-totalistic setting.

Figures 5, **7**, respectively for toroidal and spherical shapes. We observed that a certain subset of the spiking steady-state CA produced exploding patterns before reaching a randomly spiking steady-state. For any given parameter set a_0, a_1, a_2, the dynamics are observed to fall into any one of the following classes.

1. Class 0: Quiescent Steady-State: (a) Fast-decay; (b) Slow-decay.
2. Class 1: Spiking Steady-State: (a) With random patterns; (b) With exploding patterns.

The classification above becomes more obvious as we look at the steady-state trajectory shown in **Figure 6**. Here, we plotted the activity a_{t+1} vs. a_t. With spherical boundary conditions, this steady-state dynamics remains unchanged (see **Supplementary Figure 1**). Hence, the boundary condition in the systems investigated does not affect neuronal CA classification. There is a slight variation on the spatiotemporal evolution of the automaton with the spherical boundary condition as shown in **Figure 7**. The effect of spherical lattice is clearly visible on Class 1b, where the activity signal bounces back from the location of the polar-points (top and bottom rows).

In a previous work (Ramos and Bantang, 2019c), we explored the different regimes in which these CA classes exist in the phase space diagram. We found that a minimum of 20% of the population of the neurons must be active or spiking at $t = 0$ to observe a nonzero steady-state CA (Class 1). As the output threshold a_2 is increased, the systems with parameters that fall near the phase boundary, transitions from Class 0 to Class 1 CA, and thus, increasing the region for which Class

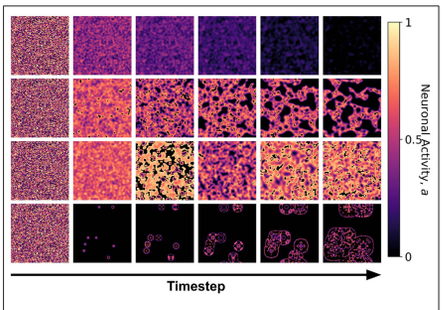


FIGURE 5 | Snapshots of the neuronal CA with toroidal lattice configuration for each class taken at different timesteps of the simulation. The snapshots are taken with increasing timesteps (not necessarily consecutive) to highlight the different dynamics observed. At $t = 0$, the initial state is the same for all classes, but each class evolved into one of the CA classes, depending on the set of parameter thresholds used in the activation function.

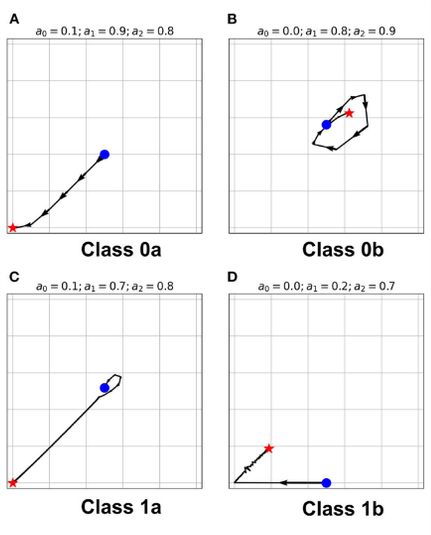


FIGURE 6 | (A–D) State-space trajectories of each of the classes. The • and ⋆ marks the automaton state at $t = 0$ and $t = \infty$, respectively. The arrows show the direction of evolution of the average neuronal steady-state over time.

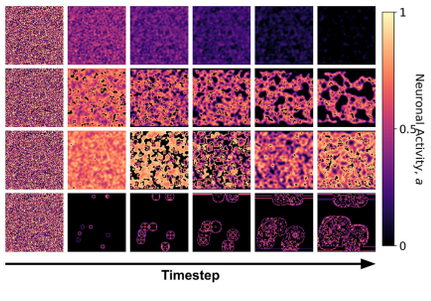


FIGURE 7 | Snapshots of the neuronal CA with spherical lattice configuration for each class taken at different timesteps of the simulation. The snapshots are taken with increasing timesteps (not necessarily consecutive) to highlight the different dynamics observed. At $t = 0$, the initial state is the same for all classes, but each class evolved into one of the CA classes, depending on the set of the parameter threshold used in the activation function.

1 CA is observed. In this work, the chosen set of parameters belong to the stable regions in which the dynamical classification is observed.

4. EFFECT OF EXTERNAL INPUT AND LAYERED LATTICE

Using the same initial state of the automaton, we assigned a certain fraction (1% and 5%) of the neurons in random locations to be c_{ext}, injected with constant $a_{\text{in}} = 1$. It is notable in **Figure 8** that for all steady-state classes, the overall system activity $\langle a \rangle$ becomes typically much greater than the input. Class0b neurons with 5% c_{ext} were found to have similar steady-state dynamics with Class 1a. Furthermore, Class 1b neurons resorted to an oscillating steady-state with 5% c_{ext}.

We implemented the method described in section 2.3 for a two-layered and four-layered CA system. The average steady-state activity remains unchanged across each layer and remains the same for the whole CA system (see **Supplementary Figure 2**). Increasing the number of layers from two to four layers did not change the CA steady-state classification. It is notable in **Figure 9** that increasing the number of layers also increases the number of neurons in the neighborhood state, and consequently delays the transition to quiescent steady-state in Class 0b. The delay is also due to the gradual decrease of the wave amplitude as it travels at least once across the system. This follows the proportionality between the system size and the time taken for the signal to propagate across the system (Wolfram, 2002).

5. COBWEB DIAGRAM ANALYSIS

Cobweb diagrams visualize how a dynamical system behaves over time (Stoop and Steeb, 2006). Consider a CA system response function defined by $a_{\text{out}} = f(a_{\text{in}})$ (see **Figure 1**). We then can draw a cobweb diagram on a plane $(x, y) = (a_{\text{in}}, a_{\text{out}})$ as follows:

1. Given a chosen starting point $(x_{\text{start}}, y_{\text{start}}) = (a_{\text{start}}, 0)$, we trace a vertical line from it to $(a_{\text{start}}, f(a_{\text{start}}))$.
2. We trace a horizontal line from $(a_{\text{start}}, f(a_{\text{start}}))$ until it crosses the dashed line with the equation $a_{\text{out}} = a_{\text{in}}$. This value becomes the new starting point, such that $(x_{\text{start}}, y_{\text{start}}) = (f(a_{\text{start}}), f(a_{\text{start}}))$.
3. Repeat steps 1 and 2 until we reach a sufficient number of steps (here, we use 100).

The resulting cobweb diagrams for the different dynamical classes are shown in **Figure 10** (Ramos and Bantang, 2020). The activation function of Class 0a falls below the line $a_{\text{out}} = a_{\text{in}}$ such that any neuron transitions to quiescent state regardless of its initial state (marked by the blue star ⋆). A collection of neurons of this class approaches a quiescent state in a short amount of timesteps. However, in Class 0b, the activation function crosses the line $a_{\text{out}} = a_{\text{in}}$ once. Any neuron state that starts from the left or right of the intersection point results in a temporary overall active state but the system eventually ends up to be in the quiescent steady-state. If the neuron state starts exactly at the intersection point, its state remains there as a trivial application of the procedure above. Only a few of neurons coincide with this trivial case since the initial state-values of the CA in our numerical experiments is obtained from a uniform random

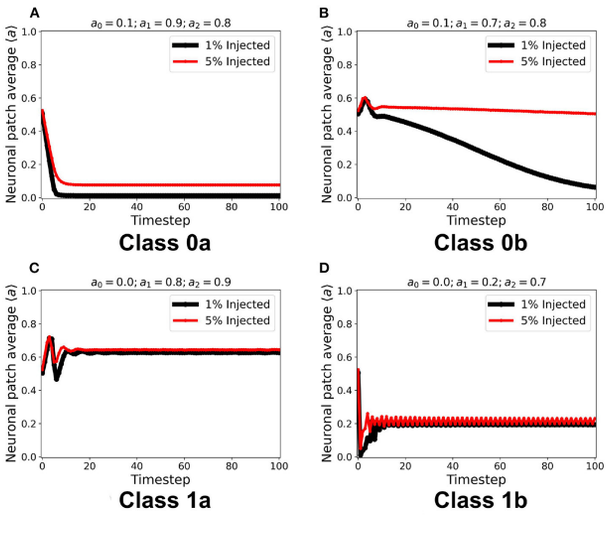


FIGURE 8 | (A–D) Steady-state dynamics for each CA class in a toroidal lattice configuration with external input injected to a fraction of the neuronal population. The black line shows the average steady-state when 1% of the population is injected with external constant input. The red line corresponds to 5% of the neuronal population injected accordingly.

distribution in the range $[0, 1]$. Collectively, Class 0b neurons go into the quiescent state but at a slower rate compared to Class 0a. Inhibitory neurons (Type I) can therefore be modeled by Class 0 neuronal patch.

If the intersection point is located at the origin (i.e. $a_0 = 0$), the collection of neurons always approaches a spiking steady-state. The greater is the difference $a_1 - a_0$, the higher the average steady-state value $\langle a \rangle$ of the system. Neuronal CAs with lower average steady-state value usually result from exploding patterns (Class 1b). Random patterns (Class 1a) consequently produce higher average steady-state values. Excitatory (Type II) neurons belong to Class 1 in the classification scheme presented.

6. BIFURCATION DIAGRAM ANALYSIS

Bifurcation diagrams show the dynamical trend of the system as we vary a parameter of interest Çelik Karaaslanl (2012). In this work, we chose to investigate the trend for varying a_2-values, holding both a_0 and a_1 at various combinations of constant values. Since 100 timesteps is enough for the simulation to reach steady-state at any given parameter set Ramos and Bantang (2018, 2019c), we obtained the average neuronal patch activity $\langle a \rangle$ only for the last (10) timesteps. The resulting bifurcation diagrams are shown in **Figure 11** (Ramos and Bantang, 2020).

At certain parameter sets, the neuronal CA exhibits period-doubling. This only happens when the activation function is negatively-sloped and strictly satisfies the conditions: $a_1 = 0$ and $a_0 > 0$. Only with this specific constraint will the overall neuron state oscillate as shown in the cobweb diagrams in **Figure 12**. An oscillating overall neuronal state indicates that a significant degree of synchronization happens in the majority fraction of the neurons. Epileptic neurons can be modeled by these negatively-sloped activation functions. This oscillatory behavior is unchanged by any neighborhood and boundary conditions, as shown in **Figure 12**. However, as we increase the fraction of neurons c_{ext} with input, the oscillation becomes underdamped.

7. EXTENDING TO NONLINEAR ACTIVATION FUNCTION

As discussed in section 1, one possible extension of the model is to consider a nonlinear activation function that provides a better approximation of the neuronal response (Hodgkin and Huxley, 1952; Gerstner et al., 2014; Pang and Bantang, 2015). The second-order approximation of the activation function is given by the equation:

$$a_{\text{out}} = \begin{cases} 0 & 0 < a_{\text{in}} < a_0 \\ a_2\left(1 - \left(1 - \frac{a_{\text{in}} - a_0}{1 - a_0}\right)^b\right) & a_{\text{in}} \geq a_0 \end{cases} \tag{2}$$

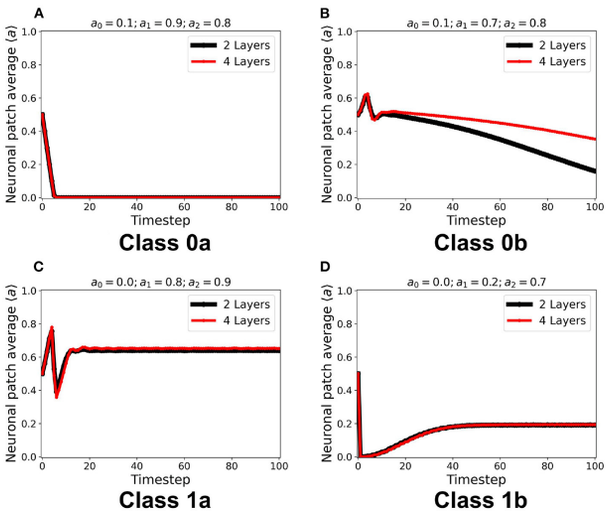


FIGURE 9 | (A–D) Steady-state dynamics of each class in a two-layered (black line) and four-layered (red line) toroidal lattice configuration. The dynamical trend remains the same for each class.

where a_0, a_2 represents the same thresholds as in Equation 1, and b is the nonlinearity parameter. Here, $a_1 = 1$. When $b = 1$, this function reverts to the first-order linear approximation. **Figure 13A** shows how the activation function changes when we increase b. An exhaustive testing of the nonlinear activation function has been done with varying input and output thresholds $a_0, a_2 \in [0, 1]$, and nonlinearity parameter $b \in [0, 40]$ (Ramos and Bantang, 2021). The resulting dynamics are classified below. Representative steady-state dynamics for each class are shown in **Figure 13B**.

1. Class 0: Quiescent Steady-State: (a) Fast-decay; (b) Slow-decay
2. Class 1: Spiking Steady-State: (a) low activation probability; (b) high activation probability.

The conditions for phase transition are presented in a previous work (Ramos and Bantang, 2021). It is notable that the opposing extreme cases ($a_2 = 0.0$ and $a_0 = 1.0$) always belong to Class 0. Whenever $0 \leq b < 1$, the system also falls under Class 0 independent of the other parameters. Once we increase the nonlinearity such that $b > 1$, especially for cases when the corresponding linear activation function lies completely in the region below the $a_{\text{out}} = a_{\text{in}}$ line, the dynamical evolution transitions from Class 0 to Class 1. This transition is caused by the crossing of the activation function to the region above $a_{\text{out}} = a_{\text{in}}$ line as shown in **Figure 13A**. Hence, using the cobweb analysis above, there are individual neurons that will contribute to an overall system spiking steady-state. We also found that, on one hand, the average steady-state activity transitions abruptly when the input threshold a_0 is decreased. On the other hand, the transition is gradual when the output threshold a_2 is increased, with the phase boundary approximated at $b \sim 1/a_2$. Furthermore, increasing the nonlinearity parameter b transitions the neuronal classification from Class 0 to Class 1.

It is notable that the opposing extreme cases ($a_2 = 0.0$ and $a_0 = 1.0$) always belong to Class 0. Whenever $0 \leq b < 1$, the system also falls under Class 0 independent of the other parameters. Once we increase the nonlinearity such that $b > 1$, especially for cases when the corresponding linear activation function lies completely in the region below the $a_{\text{out}} = a_{\text{in}}$ line, the dynamical evolution transitions from Class 0 to Class 1. This transition is caused by the crossing of the activation function to the region above $a_{\text{out}} = a_{\text{in}}$ line as shown in **Figure 13A**. Hence, using the cobweb analysis above, there are individual neurons that will contribute to an overall system spiking steady-state.

8. YOUNG AND AGED NEURONAL SYSTEMS

As an application of the proposed CA modeling paradigm, we obtained an empirical dataset of the single-cell response that shows the dynamical difference between young and aged neurons in response to input signals (Coskren et al., 2015). Data shows higher firing rates from the aged neurons. The dataset is normalized over the range of the input current ($30 - -330$pA) used in the study. When the first-order approximation activation

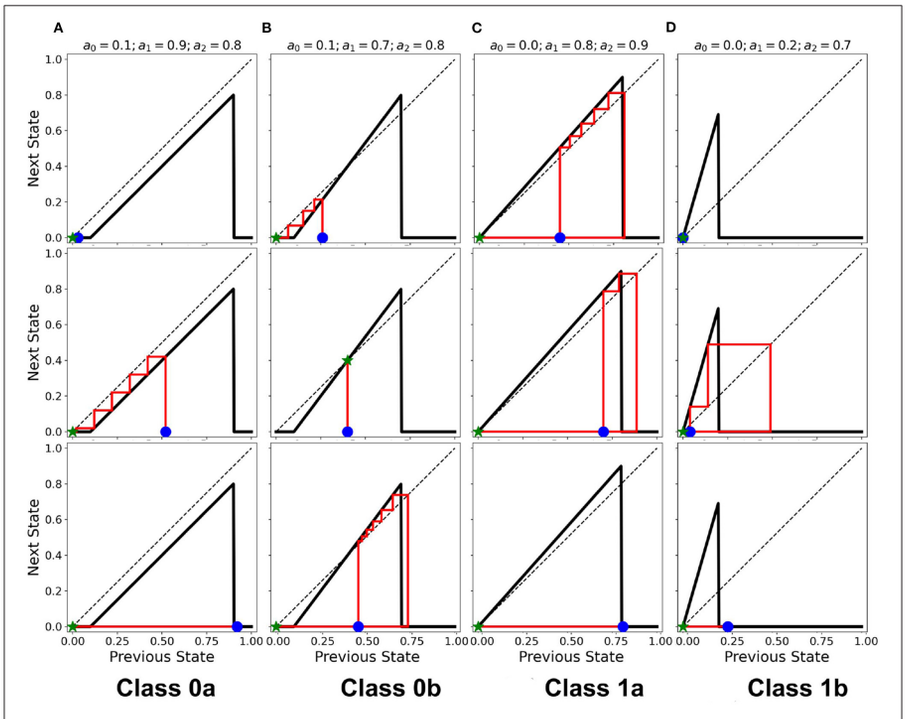


FIGURE 10 | (A–D) Cobweb diagrams showing the behavior of individual neurons for each class. In each column, the individual plots show different cases that a neuron is initialized (marked by •). Each case may not necessarily be the same across all the classes. The ⋆ marks the final state on the x-axis with the red line showing the trajectory of the neuronal state through time.

function $a_{\text{out}} = f(a_{\text{in}})$ given by Equation 1 is used to fit the dataset, both young and aged neuronal system resulted to quiescent steady-state with young neuronal CA decaying faster than the aged ones (see **Supplementary Figure 3**). If injected $a_{\text{ext}} = 1$–5% of the neuronal population, the average steady-state is the same for both young and aged neuronal systems. This result contradicts the observations by Coskren et al. (2015).

A better fitting function to the dataset is the second-order approximation given by Equation 2. **Figure 14A** shows the resulting response curve. Using this response curve, we found a significant difference in the dynamics between young and aged neuronal systems (see **Figure 14B**). The aged neuronal population shows a spiking steady-state with a higher average neuronal response than the younger population. Injecting constant external input ($a_{\text{ext}} = 1$) randomly to 5% of the neuronal population amplifies the average steady-state for both young and aged neuronal systems but with different steady-state values. Hence, the aged neuronal system does not need a very high external input for it to be amplified, unlike the younger population. This result confirms the higher firing rate of aged neurons as well as the presence of spiking states for aged at lower input currents (Coskren et al., 2015). **Figure 14C** shows the actual response of young and aged neuronal patches obtained using the method described in Ramos and Bantang (2018) and Ramos and Bantang (2021). A sample discrete response of a single neuron from the young and aged neuronal patches is shown in **Figure 14D**.

9. COMPUTATIONAL COMPLEXITY

The computational efficiency of using CA to model neuronal patch dynamics is quantified using the time it takes to finish a

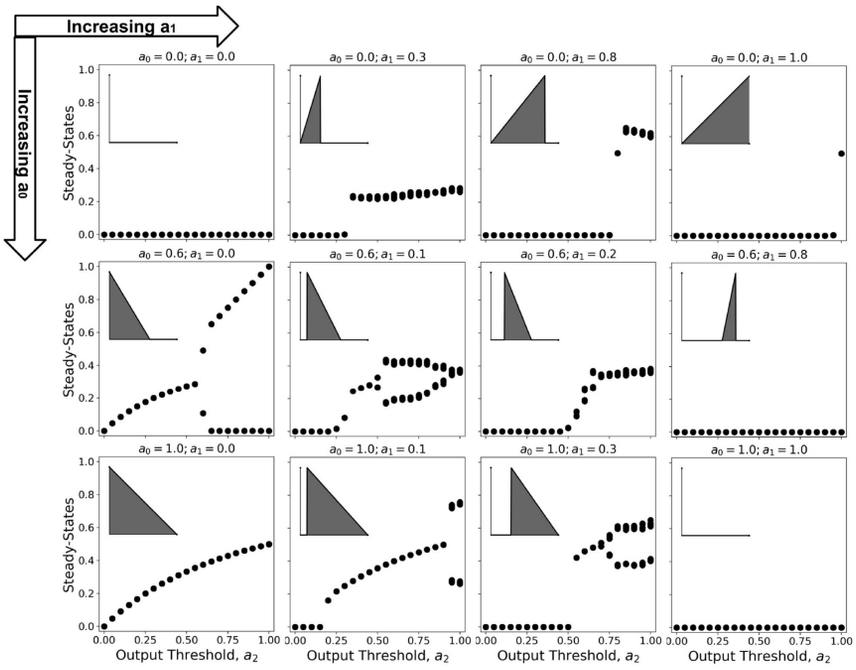


FIGURE 11 | Representative bifurcation diagrams for the neuronal CA system. Each individual plot shows the steady-state values vs. increasing a_2 with constant a_0, a_1 values. From left to right column shows the different steady-state trends with increasing a_1 while from top row to bottom shows increasing a_0. The inset shows the activation function for that parameter set given that $a_2 = 1.0$, for reference.

given simulation. With increasing neuronal population N, the time it takes to finish the simulation T is recorded as the average of three (3) runs or trials for each solver. In general, we find that the time T can be fitted with:

$$T = a(N - b)^c \tag{3}$$

where a, b and c are the fitting parameters. From these parameters, c provides the relevant information for the computational complexity whereby $T \sim O(N^c)$ for large N-values.

Figure 15 shows the comparison of the simulation time using different solvers to the HH ODEs and the CA modeling method presented here. On one hand, solving the ODEs of the HH neuronal network using the forward Euler method yields quadratic time complexity ($T \sim O(N^2)$). Using more accurate solver variants such as Runge-Kutta order-4 (RK4) and Livermore Solver for Ordinary Differential Equations(LSODA) increases the overall magnitude of T yet returns consistent complexity $c \approx 2$. On the other hand, our CA model presents a linear time complexity ($T \sim O(N)$) indicating a much faster computational time than simulating interconnected HH neurons, especially for much larger system size N.

Figure 15 also shows that the T-values for HH neuronal population sizes beyond $N = 4,096$ is absent. Running simulations for larger sizes causes our current computational machines to exceed their memory capacity. The use of our CA model shows significant advantage in simulating up to millions of neurons (more than 10^3 times the other reported approach) before this memory problem happens. The algorithm of the CA model can be more straightforwardly parallelized and GPU-implemented to amplify the neuronal population without increasing the simulation time.

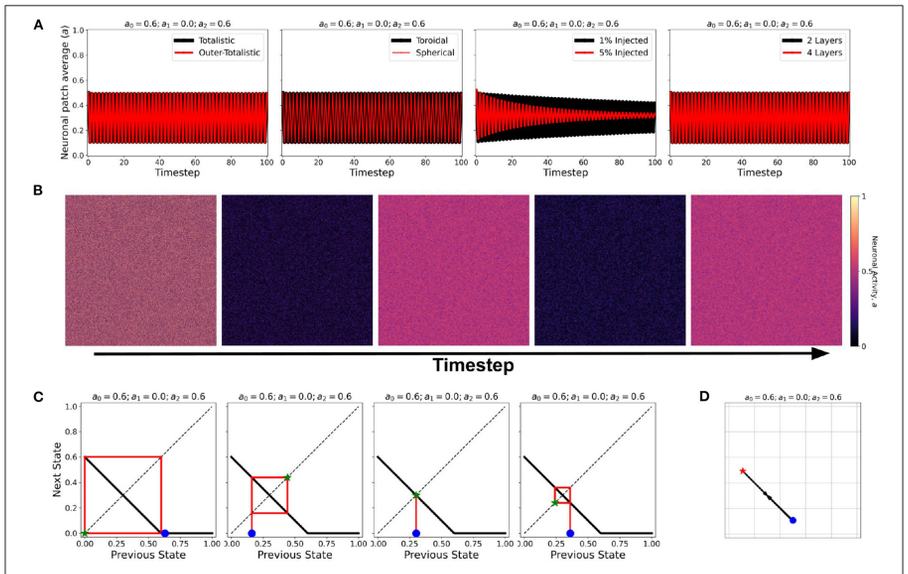


FIGURE 12 | (A) Steady-state dynamics of Class 2 CA using the various neighborhood and boundary conditions aforementioned. **(B)** Snapshots showing spatiotemporal evolution of Class 2 CA. **(C)** Cobweb diagrams for each possible initial condition (marked by •) in a Class 2 CA with threshold values $a_0 = 0.6$, $a_1 = 0.0$, and $a_2 = 0.6$. **(D)** Trajectory of the neuronal steady-state activity in state-space. The average state of the CA are marked at $t = 0$ (•) and at $t = \infty$ (•) with the black arrows tracing the trajectory. It is notable that the neuronal CA cycles back and forth along the path of evolution.

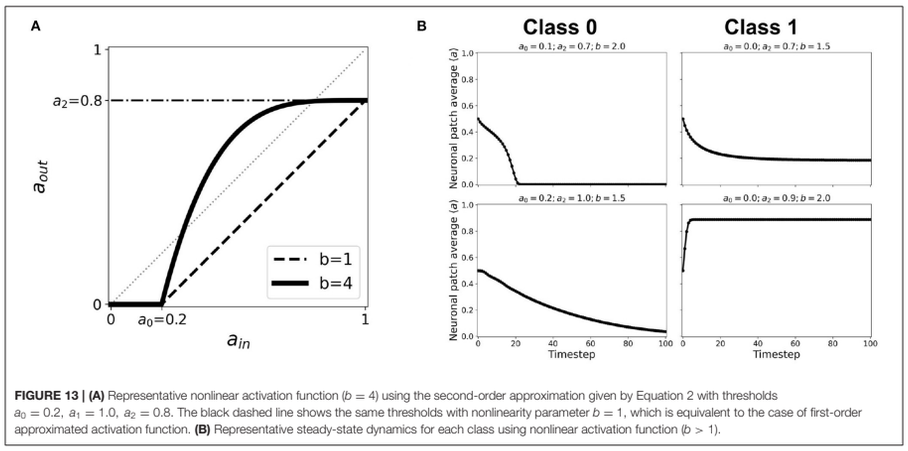


FIGURE 13 | (A) Representative nonlinear activation function ($b = 4$) using the second-order approximation given by Equation 2 with thresholds $a_0 = 0.2$, $a_1 = 1.0$, $a_2 = 0.8$. The black dashed line shows the same thresholds with nonlinearity parameter $b = 1$, which is equivalent to the case of first-order approximated activation function. **(B)** Representative steady-state dynamics for each class using nonlinear activation function ($b > 1$).

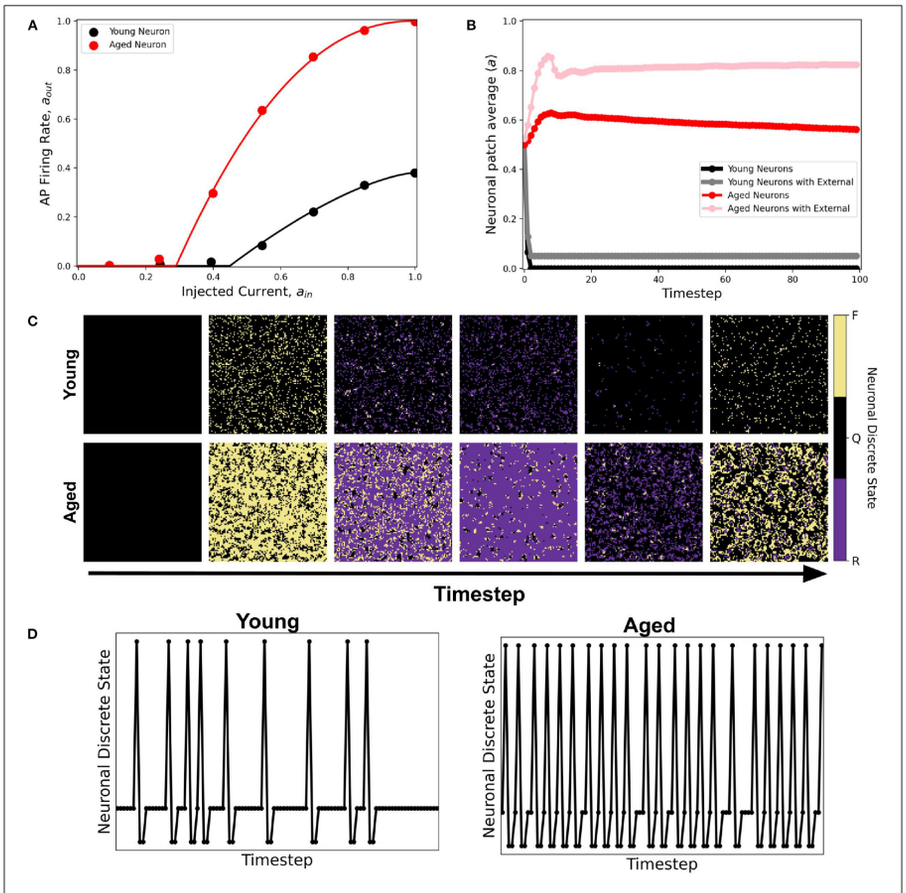


FIGURE 14 | (A) Activation function of young (black) and aged (red) neuronal population derived from empirical data from rhesus monkey prefrontal cortex (Coskren et al., 2015). The dots are the dataset from the study, while the solid lines are the activation function fitted using Equation 2. The parameter thresholds are $a_0 = 0.45$, $a_2 = 0.38$, $b = 1.5$ for the young neuronal system, and $a_0 = 0.29$, $a_2 = 1.0$, $b = 2.2$ for the aged population. **(B)** Corresponding average steady-state dynamics of young and aged neuronal population using the CA model. **(C)** Actual response of young and aged neuronal patch obtained using the method described in Ramos and Bantang (2018) and Ramos and Bantang (2021). The color represents the discrete state the neuron is in: Q (Quiescent), F (Firing), R (Refractory). **(D)** Representative temporal response of a single neuron from the young and aged neuronal patch according to the discrete states in **(C)**.

10. CONCLUSIONS

In this work, we proposed a cellular automata (CA) model as an efficient way of modeling large numbers of neurons that can reduce both computational time and memory requirements in simulation. We implemented neuronal dynamics on a neuronal CA patch of lattice size $1,024 \times 1,024$ using a first-order linear approximation of the resulting activation function of the HH model. The system dynamics is characterized according to the three parameters of the resulting activation function.

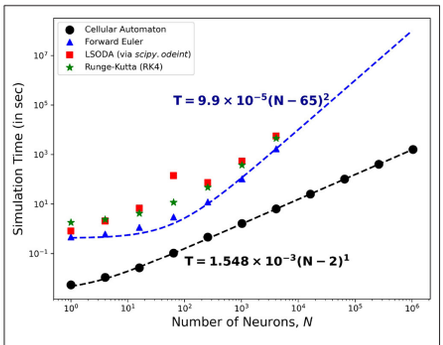


FIGURE 15 | Time complexity of the neuronal model comparing different algorithms for solving the HH dynamics. The forward Euler method (marked ▲) shows quadratic order in time (∼$O(N^2)$) while the CA model (marked •) presents a linear complexity time (∼$O(N)$) indicating the efficiency of using the CA model in simulation of neuronal patch dynamics.

The steady-state dynamics are investigated for different lattice configuration (2D and quasi-3D), boundary conditions (toroidal and spherical), layering (one- or two-layered), and Moore neighborhood type (totalistic and outer-totalistic). Cases wherein a fraction (1% and 5%) of neurons have constant activation input (a_{in}) are also explored. We observed the following CA classification:

1. Class 0: Quiescent Steady-State: (a) Fast-decay; (b) Slow-decay
2. Class 1: Spiking Steady-State: (a) With random patterns; (b) With exploding patterns
3. Class 2: Oscillatory Steady-State.

Numerical experiments of CA neuronal systems are shown to conform to this classification. While our analysis of the cobweb diagrams show that individual neuron states will eventually reduce to quiescent state, spiking steady-state can still emerge for a collection of interconnected neurons. Collective oscillatory behavior (Class 2) of the overall neuronal state is observed for the system with significant synchronization among neurons.

The proposed CA model is applied to analyze the resulting dynamical class of young and aged patches of neurons. The response function of individual young and aged neuronal cells are obtained from empirical data and are fitted to a second-order approximation for better semblance. The CA model for aged patch shows dynamics with higher average neuronal steady-state and therefore more robust spiking behavior compared to the younger population. This result conforms to the higher action potential firing rates of aged neurons from the empirical data. On one hand, the average neuronal steady-state is amplified for the aged population when injected with a small external input. On the other hand, the younger population needs higher external input to observe significant amplification of the average neuronal steady-state. This result conforms to the presence of spiking activity in aged neurons stimulated with lower external current. Whether artificially generated spatiotemporal patterns of neuronal patch activity in this work correspond to the activity of actual neuronal systems remains to be determined.

The cellular automata model presented here can easily be extended to model more realistic neuronal systems such as brain patches or even the whole brain. Individual neuronal response data can also be used to improve the choice of the CA activation function $a_{\text{out}} = f(a_{\text{in}})$. The activation function can be modified into similar input-output mapping in frequency domain or voltage-current domain, and can be used as the rule for our CA model. Our computational modeling framework can be utilized for large scale simulation of different neuronal conditions such as Parkinson's disease (Bevan et al., 2002; Kang and Lowery, 2014), Alzheimer's disease, and chronic traumatic encephalopathy (Gabrieli et al., 2020; Wickramaratne et al., 2020).

We presented here that an adult brain shows an increase of neuronal response, even in the presence of constant external input. However, it remains a challenge to understand in which particular biological aspect these changes correspond to. In future studies, we recommend investigating dynamical systems of interconnected neurons, both young and aged, in the following aspects: 1) input-output mapping; 2) spatiotemporal distribution; and 3) connectivity architecture. Learning about the dynamics of these systems would help medical practitioners to detect early signs of ailments or disorders stemming from the aging process and help identify appropriate medicinal (chemical, radiation), behavioral (lifestyle, dietary) and/or surgical intervention.

DATA AVAILABILITY STATEMENT

The original contributions presented in the study are included in the article/**Supplementary Material**, further inquiries can be directed to the corresponding author.

AUTHOR CONTRIBUTIONS

RR did the simulation and wrote the manuscript draft. All authors conceived the research problem and analyzed the results, and edited the final manuscript.

ACKNOWLEDGMENTS

This work was supported by the Institute for Neurosciences, St. Luke's Medical Center. RXAR acknowledges the Department of Science and Technology (DOST) for his Advanced Science and Technology Human Resources Development Program (ASTHRDP) scholarship.

SUPPLEMENTARY MATERIAL

The Supplementary Material for this article can be found online at: https://www.frontiersin.org/articles/10.3389/fninf.2021.763560/full#supplementary-material

REFERENCES

Ananthanarayanan, R., Esser, S. K., Simon, H. D., and Modha, D. S. (2009). "The cat is out of the bag: Cortical simulations with 109 neurons, 1013 synapses," in *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis, SC '09* (New York, NY: Association for Computing Machinery), 1–12.

Arciaga, M., Pastor, M., Batac, R., Bantang, J., and Monterola, C. (2009). Experimental observation and an empirical model of enhanced heap stability resulting from the mixing of granular materials. *J. Stat. Mech.* 2009, P07040. doi: 10.1088/1742-5468/2009/07/P07040

Bevan, M. D., Magill, P. J., Hallworth, N. E., Bolam, J. P., and Wilson, C. J. (2002). Regulation of the timing and pattern of action potential generation in rat subthalamic neurons *in vitro* by gaba-a ipsps. *J. Neurophysiol.* 87, 1348–1362. doi: 10.1152/jn.00582.2001

Brunel, N., and Van Rossum, M. C. (2007). Quantitative investigations of electrical nerve excitation treated as polarization. *Biol. Cybern.* 97, 341–349. doi: 10.1007/s00422-007-0189-6

Çelik Karaaslanl, C. (2012). "Bifurcation analysis and its applications," in *Numerical Simulation: From Theory to Industry, Chapter 1*, ed M. Andriychuk (London: INTECH Open Access Publisher, Pidstryhach Institute for Applied Problems of Mechanics and Mathematics), 3–34.

Coskren, P. J., Luebke, J. I., Kabaso, D., Wearne, S. L., Yadav, A., Rumbell, T., et al. (2015). Functional consequences of age-related morphologic changes to pyramidal neurons of the rhesus monkey prefrontal cortex. *J. Comput. Neurosci.* 38, 263–283. doi: 10.1007/s10827-014-0541-5

Dalton, J. C., and FitzHugh, R. (1960). Applicability of Hodgkin-Huxley model to experimental data from the giant axon of lobster. *Science* 131, 1533–1534. doi: 10.1126/science.131.3412.1533

Gabrieli, D., Schumm, S. N., Vigilante, N. F., Parvesse, B., and Meaney, D. F. (2020). Neurodegeneration exposes firing rate dependent effects on oscillation dynamics in computational neural networks. *PLoS ONE* 15:e234749. doi: 10.1371/journal.pone.0234749

Gerstner, W., Kistler, W. M., Naud, R., and Paninski, L. (2014). *Neuronal Dynamics: From Single Neurons to Networks and Models of Cognition and Beyond*. Cambridge: Cambridge University Press.

Hawick, K., and Scogings, C. (2011). "Cycles, transients, and complexity in the game of death spatial automaton," in *Proceedings of International Conference on Scientific Computing (CSC-11)*. (Las Vegas, NV: CSREA), 241–247.

Hodgkin, A. L., and Huxley, A. F. (1952). A quantitative description of membrane current and its application to conduction and excitation in nerve. *J. Physiol.* 117, 500–544. doi: 10.1113/jphysiol.1952.sp004764

Kang, G., and Lowery, M. (2014). Effects of antidromic and orthodromic activation of stn afferent axons during dbs in parkinson's disease: a simulation study. *Front. Comput. Neurosci.* 8:32. doi: 10.3389/fncom.2014.00032

Lapicque, L. (1907). Recherches quantitatives sur l'excitation électrique des nerfs traitée comme une polarisation. *J. Physiol. Pathol. Gen.* 9, 620–635.

Morris, C., and Lecar, H. (1981). Voltage oscillations in the barnacle giant muscle fiber. *Biophys. J.* 35, 193–213. doi: 10.1016/S0006-3495(81)84782-0

Mostany, R., Anstey, J. E., Crump, K. L., Maco, B., Knott, G., and Portera-Cailliau, C. (2013). Altered synaptic dynamics during normal brain aging. *J. Neurosci.* 33, 4094–4104. doi: 10.1523/JNEUROSCI.4825-12.2013

Nagumo, J., Arimoto, S., and Yoshizawa, S. (1962). An active pulse transmission line simulating nerve axon. *Proc. IRE* 50, 2061–2070. doi: 10.1109/JRPROC.1962.288235

Noble, D. (1962). A modification of the Hodgkin Huxley equations applicable to Purkinje fibre action and pacemaker potentials. *J. Physiol.* 160, 317–352. doi: 10.1113/jphysiol.1962.sp006849

Pang, J. C. S., and Bantang, J. Y. (2015). Hodgkin huxley neurons with defective and blocked ion channels. *Int. J. Modern Phys. C* 26, 1550112. doi: 10.1142/S0129183115501120

Pannese, E. (2011). Morphological changes in nerve cells during normal aging. *Brain Struct. Funct.* 216, 85–89. doi: 10.1007/s00429-011-0308-y

Peters, A. (2007). "The effects of normal aging on nerve fibers and neuroglia in the central nervous system," in *Brain Aging*. (Boca Raton, FL: CRC Press), 97–126.

Ramos, R., and Bantang, J. (2018). "Proposed cellular automaton model for a neuronal patch with a thresholded linear activation function," in *Proceedings of the Samahang Pisika ng Pilipinas, Vol. 36* (SPP-2018-PB-37), (Puerto Princesa).

Ramos, R., and Bantang, J. (2019a). "Classification of the dynamics of an outer-totalistic 2D and quasi-3D cellular automata simplistic models of neuronal patches," in *16th International Conference on Molecular Systems Biology* (Manila: BST 2019, De La Salle University Poster presentation), Available online at: https://drive.google.com/file/d/1bs655p68uv0SpZmmOcDr8MFV-_bWLvo_/view.

Ramos, R., and Bantang, J. (2019b). "An outer-totalistic 2D and quasi-3D cellular automata simplistic models of neuronal patches," in *Brain Connects 2019 and the 9th Neuroscience International Symposium* (Taguig: St. Luke's Medical Center, Global City. Poster and oral presentation), BC2019-14.

Ramos, R. X., and Bantang, J. (2019c). "Totalistic cellular automata model of a neuronal network on a spherical surface," in *Proceedings of the Samahang Pisika ng Pilipinas, Vol. 37*, (Tagbilaran). SPP-2019-PB-16.

Ramos, R. X. A. (2019). *Dynamics of a neuronal lattice network with a linear activation function using cellular automata modelling* (Bachelor's thesis). National Institute of Physics, College of Science, University of the Philippines, Diliman, Quezon City.

Ramos, R. X. A., and Bantang, J. Y. (2020). "Verhulst and bifurcation analyses of a neuronal network on an outer-totalistic toroidal cellular automata," in *Proceedings of the Samahang Pisika ng Pilipinas, Vol. 38*, (Quezon City). SPP-2020-3C-05.

Ramos, R. X. A., and Bantang, J. Y. (2021). "Simplified cellular automata model of neuronal patch dynamics with generalized non-linear cell response," in *Proceedings of the Samahang Pisika ng Pilipinas, Vol. 39*, (Quezon City). (SPP-2021-PB-03).

Rivera, A. D., Pieropan, F., Chacon-De-La-Rocha, I., Lecca, D., Abbracchio, M. P., Azim, K., et al. (2021). Functional genomic analyses highlight a shift in gpr17-regulated cellular processes in oligodendrocyte progenitor cells and underlying myelin dysregulation in the aged mouse cerebrum. *Aging Cell.* 20:e13335. doi: 10.1111/acel.13335

Stoop, R., and Steeb, W.-H. (2006). *Berechenbares Chaos in Dynamischen Systemen*. Basel: Springer-Verlag.

Tal, D., and Schwartz, E. (1997). Computing with the leaky integrate-and-fire neuron: logarithmic computation and multiplication. *Neural Comput.* 9, 305–318. doi: 10.1162/neco.1997.9.2.305

von Neumann, J. (1966). *Theory of Self-Reproducing Automata*. Champaign, IL: University of Illinois Press.

Wickramaratne, S. D., Mahmud, M. S., and Ross, R. S. (2020). "Use of brain electrical activity to classify people with concussion: a deep learning approach," in *ICC 2020-2020 IEEE International Conference on Communications (ICC)* (Dublin), 1–6.

Wilson, H. R., and Cowan, J. D. (1972). Excitatory and inhibitory interactions in localized populations of model neurons. *Biophys. J.* 12, 1–24. doi: 10.1016/S0006-3495(72)86068-5

Wolfram, S. (2002). *A New Kind of Science*. Champaign, IL: Wolfram Media, Inc. Available online at: https://www.wolframscience.com/nks/.

Conflict of Interest: The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

frontiers

Supplementary Material

1 SUPPLEMENTARY FIGURES

1.1 Toroidal CA vs Spherical CA

Figure S1 compares the average neuronal state between toroidal CA and spherical CA. The steady-state remains unchanged indicating that the proposed neuronal classification scheme holds for both CA lattice configurations.

1.2 Two-Layered Lattice Dynamics

The average neuronal state for each layer in a two-layered lattice CA is shown in Figure S2. For each class, the neuronal dynamics remains unchanged for all layers and is also the same with the average of the whole layered lattice system.

1.3 Young vs Aged Neuronal Population with Linear Activation Function

Figure S3A shows the activation function of young and aged neuronal population using an empirical dataset of rhesus monkey prefrontal cortex (Coskren et al., 2015). The dataset is fitted using the following equation:

$$a_{\text{out}} = \begin{cases} 0 & a_{\text{in}} < a_0 \\ \frac{a_2(a_{\text{in}}-a_0)}{a_1-a_0} & a_{\text{in}} \in [a_0, a_1] \\ 0 & a_{\text{in}} > a_1 \end{cases} \tag{S1}$$

The resulting dynamics (see Figure S3B) shows similar steady-state behavior for both young and aged population. When subjected with external input, both young and aged population amplifies to the same average steady-state value. These results is contrary to what was observed by Coskren et al. (2015). Hence, the activation function is modified into the second-order approximation for better semblance.

2 SUPPLEMENTARY VIDEO ANIMATIONS

The supplementary movie files (mp4 format) show the evolution of 1024×1024 neuronal lattice with an outer-totalistic toroidal configuration for each classes.

SuppFigure4. Spatiotemporal Evolution of Class 0a CA

SuppFigure5. Spatiotemporal Evolution of Class 0b CA

SuppFigure6. Spatiotemporal Evolution of Class 1a CA

SuppFigure7. Spatiotemporal Evolution of Class 1b CA

SuppFigure8. Spatiotemporal Evolution of Class 2 CA

REFERENCES

Coskren PJ, Luebke JI, Kabaso D, Wearne SL, Yadav A, Rumbell T, et al. Functional consequences of age-related morphologic changes to pyramidal neurons of the rhesus monkey prefrontal cortex. *Journal of computational neuroscience* **38** (2015) 263–283. doi:10.1007/s10827-014-0541-5.

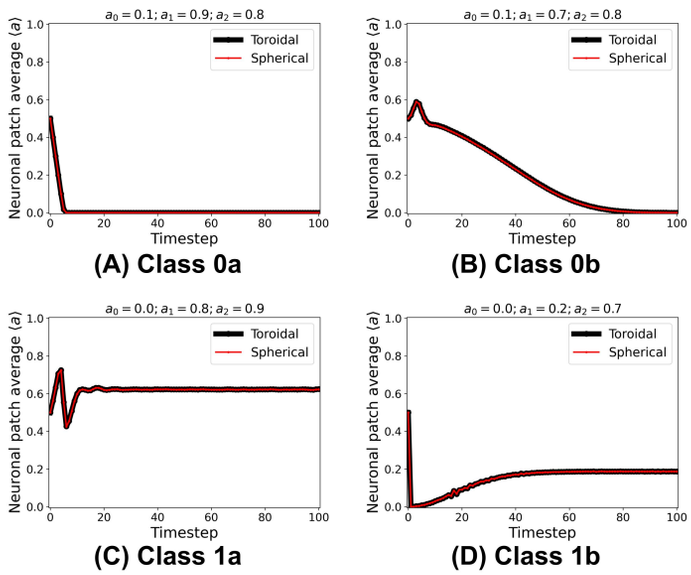


Figure S1. Steady-state dynamics of each class in a toroidal (black line) and spherical (red line) lattice configuration. The dynamical trend remains the same for each class.

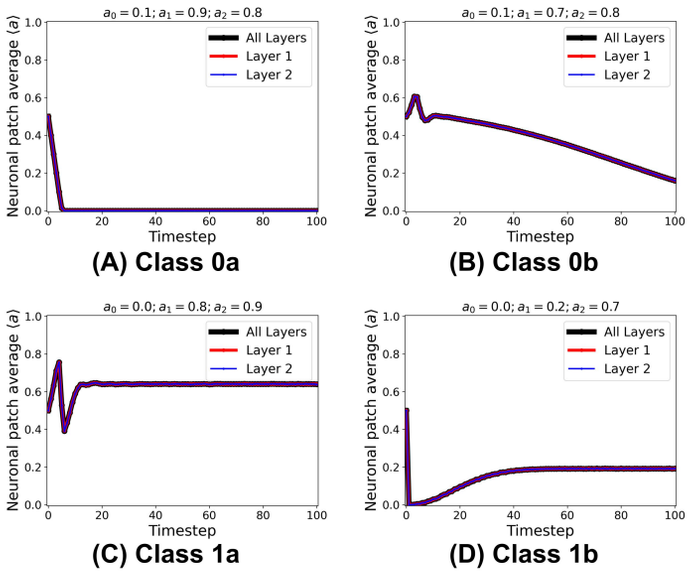


Figure S2. Steady-state dynamics of each class in a two-layered toroidal lattice. The average neuronal activity is the same across each layer and has similar dynamics with the average activity of the whole CA system.

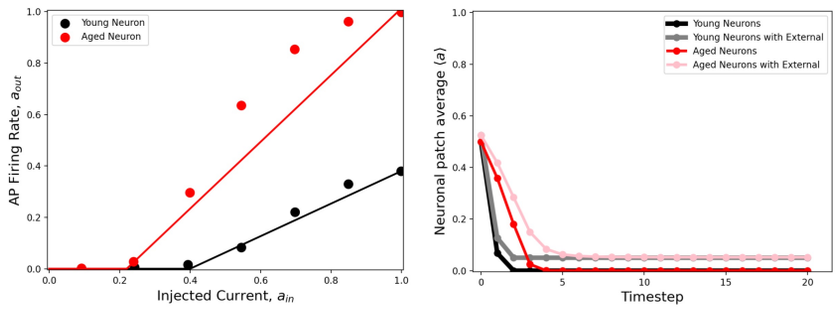


Figure S3. **(A)** Activation function of young (black) and aged (red) neuronal population derived from empirical data from rhesus monkey prefrontal cortex (Coskren et al., 2015). The dots are the dataset from the study, while the solid lines are the activation function fitted using the linear activation function Equation S1. The parameter thresholds are $a_0 = 0.4,\ a_1 = 1.0,\ a_2 = 0.38$ for the young neuronal system, and $a_0 = 0.22,\ a_1 = 1.0,\ a_2 = 1.0$ for the aged population. **(B)** Corresponding average steady-state dynamics of young and aged neuronal population using the CA model.

Simplified cellular automata model of neuronal patch dynamics with generalized non-linear cell response

Reinier Xander A. Ramos* and Johnrob Y. Bantang

National Institute of Physics, University of the Philippines Diliman, Quezon City, Philippines
*Corresponding author: rramos@nip.upd.edu.ph

Abstract

Hodgkin-Huxley (HH) and other computational neuronal models are adequate methods in describing the dynamics of a single neuron. However, the HH model consist of four ordinary differential equations (ODEs) to solve for the neuronal response. Hence, simulating a large network of HH neurons would require very powerful computing devices. A model based on cellular automata (CA) has been recently made possible resulting to a simplistic way of simulating many neurons without requiring large computational resource. In this study, we introduce a non-linear CA that generalizes the response of the neuron compatible with empirical data via a nonlinearity parameter b. The resulting simple model is used to study a neuronal patch of 10^4 ($N = 10\ 000$) neurons in a 2D lattice with periodic boundary conditions. The phase space diagrams show that in the limit that the activation function becomes linear ($b \to 1^+$), the system transitions from an active steady-state (Class 1) to a quiescent steady-state (Class 0) at $a_0 \approx 0.5$. Actual spatio-temporal neuronal responses simulations are obtained by translating the activation probability into dynamical transitions between the three standard neuronal cell (discrete) states: 1) Q = quiescent or inactive; 2) F = firing or spiking; 3) R = refractory period, such that Q $\to$ F $\to$ R $\to$ Q.

Keywords: Models of single neurons and networks (87.19.ll), Neuronal network dynamics (87.19.lj)

1 Introduction

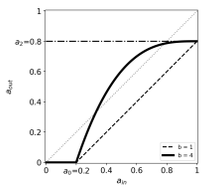
Throughout the last century, many neuronal models have since improved the integrate-and-fire model developed by Louis Lapicque in 1907 [1]. The advent of advanced technology enhanced the extent of the computational neuronal models into solving ordinary differential equations (ODEs) that describe the response of a single neuron. One of these neuronal models is the Hodgkin-Huxley (HH) which uses four (4) coupled ODEs to express how action potentials in the neuron are initiated and propagated [2, 3]. The application of the model to a squid giant axon earned Alan Hodgkin and Andrew Huxley a Nobel Prize in Physiology or Medicine in 1963.

When the HH neurons are interconnected into a network, the computation, both manual and analytical, tends to increase rapidly [4]. Hence, implementation of a large network of HH neurons requires a more powerful computing device. One way to overcome this difficulty is to take an approximation of the HH response. In a previous work, a simplified linear approximation of the HH response is used as the update rule in collection of neurons via cellular automata (CA) model [5]. We extended to a nonlinear approximation for this study incorporating the properties observed in a typical HH neuron: a) an input threshold in which the neuron fires above this threshold, b) a maximum threshold for the firing rate, and c) the firing rate monotonically increasing above the input threshold. These properties distinguish between the HH neuronal response and the activation function in neural networks (not to be confused with neuronal networks).

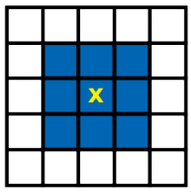
The cellular automata model enables us to simulate up to thousands of interconnected neurons as if the network is a neuronal tissue or patch [6]. In this study, a second-level approximation of the HH response (Figure 1a) is applied on an outer-totalistic CA model. This approximation considers the nonlinearity of the neuronal response function, and hence provides a better approximation to the response of the biological neuron. Our activation function does not directly correspond to the actual neuronal response over time, but denotes the probability of firing of the neuron at that specific time.

2 Neuronal CA Model

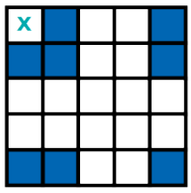
The model consists of 10^4 neurons arranged in a 100×100 two-dimensional (2D) lattice with periodic or toroidal boundary conditions. Larger neuronal population can be simulated given the availability of GPU-computing devices. At the start of the simulation, each cell in the lattice is assigned with a random probability a taken from a uniform distribution from $[0, 1]$. The connections of each cell is defined by an



(a) activation function



(b) outer-totalistic Moore neighborhood



(c) toroidal boundary

Figure 1: a) Sample activation function with thresholds $a_0 = 0.2$ and $a_2 = 0.8$. The solid line shows the second-level approximation with $b = 4$ as compared for the linear case $b = 1$ (dashed line). The gray dotted line represents the $a_{\text{out}} = a_{\text{in}}$ line. b-c) Neighborhood and boundary condition used in the neuronal CA model. In an outer-totalistic Moore neighborhood, the cell 'X' is connected to itself and to the adjacent cells in all the directions around it. Under the toroidal or periodic boundary condition, the cells at the edges are connected on the opposite edge of the lattice so that the number of neighbors of the cell will be uniform.

outer-totalistic Moore neighborhood [6], described in Figures 1b and 1c. At each timestep, the average a-values of the neighborhood is recorded as the input a_{in} and is mapped into the nonlinear activation function given by:

$$a_{\text{out}} = \begin{cases} 0 & 0 < a_{\text{in}} < a_0 \\ a_2 \left[1 - \left(\frac{1 - a_{\text{in}}}{1 - a_0} \right)^b \right] & a_0 \leq a_{\text{in}} \leq 1 \end{cases} \tag{1}$$

where a_0 and a_2 are the respective input and output cut-offs, and b is the nonlinearity parameter. This activation function is essentially a "return map" of each neuron consistent with empirical data[2, 4]. The state of the neuron at the next timestep is determined by the output a_{out}.

To approximate the actual neuronal response (each CA cell), we define three discrete states: 1) Q : quiescent or inactive; 2) F : firing or spiking; 3) R : refractory. These three states are derived from the same states in Brian's brain automaton [7]. The neuronal patch is initialized to be all quiescent state. At each timestep, if the previous neuronal state is Q, then the neuron has probability of $\langle a \rangle_{\mathcal{N}}$ (averaged over the neighboring cells $\mathcal{N}$) to fire (F) at the next timestep. All neurons in state F always automatically transition to state R (one timestep) and remains in R for the next two timesteps before it automatically transitions back to state Q. This transition rule is summarized in Figure 2a. The duration of the transition rules is chosen phenomenologically to match that of the biological neuron (1 ms for F-to-R, 2 ms for R-to-Q) [8]. This is an approximation of the actual response of the neuron if we consider 1 simulation timestep = 1 ms in real-time.

3 Numerical Experiments

Our numerical simulations consist of 10^4 ($N = 10\,000$) neurons. The simulation duration is 100 timesteps. We performed exhaustive testing of the activation function (Equation 1) by varying the input and output thresholds $a_0, a_2 \in [0, 1]$ in intervals of 0.025, and nonlinearity parameter $b \in [0, 40]$ with interval 0.1. The average of the last ten (10) timesteps are recorded for each simulation and taken as the steady-state. These values are shown in the phase space in Figure 2b for the extreme cases $a_2 = 1.0$ and $a_0 = 0.0$. On one hand, the input threshold phase space shows mostly two average steady-state values (0 and 1). This forms the basis for neuronal CA classification into two: 1) Class 0: Quiescent Steady-State; and 2) Class 1: Spiking Steady-State. On the other hand, the output threshold phase space shows the full spectrum of average steady-state values. As the output threshold a_2 increases, the average steady-state value also increases. Furthermore, increasing the nonlinearity parameter b transitions the neuronal classification from Class 0 to Class 1. This is more apparent when we compare the steady-state values for the case of $b = 2$ versus $b = 10$, where the Class 1 region increases with b. The phase boundary is approximated at $b \approx \frac{1}{a^2}$. Furthermore, it is notable that the opposite extreme cases ($a_2 = 0.0$ and $a_0 = 1.0$) always have zero steady-state (Class 0 in our scheme). The case when $0 < b < 1$ (not shown in the phase space diagram) also falls under Class 0 neuronal CA.

Figure 3 shows representative steady-state dynamics for each neuronal CA class. Class 0 neuronal CA shows two types of temporal behaviors: a) a fast decay zero steady-state; and b) slow-decay zero

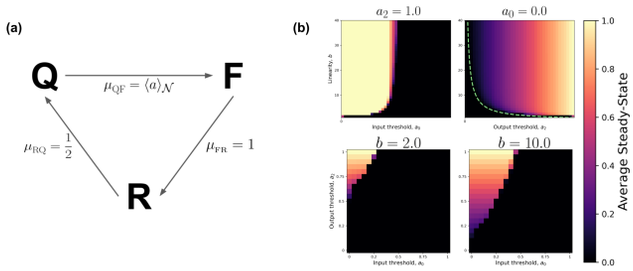


Figure 2: **(a)** Neuronal state transition rules using discrete states derived from the Brian's brain automaton. The neuron in firing (F) state always undergo a refractory period (R) for two timesteps. An R state always transition to quiescent (Q). An inactive or quiescent (Q) neuron has a probability to fire (F) with the probability $\langle a \rangle_{\mathrm{neighbors}}$. **(b)** Phase space for the cases $a_2 = 1.0$, $a_0 = 0.0$, $b = 2.0$, and $b = 10.0$ showing the phase transition from Class 0 to Class 1 neuronal CA. The color represents the average steady-state for the last ten timesteps of the simulation. The green dashed line in the second plot shows the phase boundary $b = 1/a_2$.

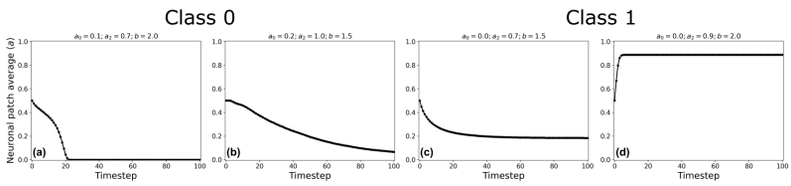


Figure 3: Representative steady-state dynamics for the neuronal CA. From an exhaustive search given the range of the parameters, the simulation usually takes up to 30 timesteps to achieve steady-state except for **(b)** which takes up to 200 timesteps to achieve quiescent steady-state. Two temporal behaviors are observed for Class 0: **(a)** slow-decay, and **(b)** fast-decay. The representative plots for Class 1 CA show **(c)** low steady-state activation probability, and **(d)** high steady-state activation probability.

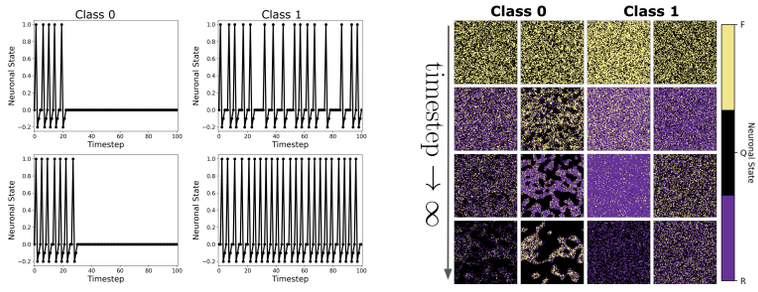
steady-state. The representative plots for Class 1 CA show a) a low activation probability, and b) a high activation probability.

From the obtained activation probabilities a, we applied the discrete neuronal state transition rules outlined in Figure 2a. The resulting temporal dynamics of a single neuron are shown in Figure 4a. The representative plots shown here follow the same parameters as those in Figure 3. The slow-decaying Class 0 shows fewer spikes as compared to the fast-decaying type. Both types, however, show a quiescent steady-state which is analogous to the behavior of inhibitory neurons in the absence of external input [9]. In our model, the neurons in Class 0 CA tend to inhibit their neighbors, thus lowering the average firing probability. In Class 1 CA, shorter spike trains were generated from the lower steady-state activation probability as compared from higher steady-state activation probability that shows longer spike trains. The spike trains are self-sustained over time even in the absence of input external to the system. This characteristic is exhibited by excitatory neurons [9].

The aforementioned temporal behavior is reflected on the spatiotemporal evolution of the automata. Figure 4b shows the snapshots of the evolution of the neuronal patch taken at various timesteps over the simulation duration. In Class 0 neuronal CA, the neuronal activity diffuses from the activation sites but are not enough to sustain the activity across the automaton. This contributes to the decay of the neuronal signal leading to a quiescent steady-state. In Class 1 neuronal CA, the neuronal activity is self-sustained over time. Fewer activation sites are observed for Class 1 with lower steady-state activation probability.

4 Conclusions

As a summary, we take the second level approximation of the response of the Hodgkin-Huxley neuron. The simplified activation function was implemented as the rule for a cellular automaton model consisting of 10^4 cells arrange in a 100×100 two-dimensional lattice. The resulting steady-state dynamics of the



(a) single neuron response (b) CA spatiotemporal evolution

Figure 4: a) Representative temporal dynamics for a single neuron response using the discrete states. Each neuronal state is assigned an output state value for visualization purposes: ($Q = 0$, $F = 1$, $R = -0.2$). b) Snapshots of the neuronal CA evolution taken at various timesteps over the simulation duration. The frames are taken at different sequential (not necessarily consecutive) timesteps to highlight the over-all behavior of the neuronal patch. The color represents the discrete neuronal state (Q, F, R).

neuronal patch response are classified into two: 1) Class 0 — Quiescent Steady-State, and 2) Class 1 — Spiking Steady-State. The steady-state activation probability gradually transitions from Class 0 to Class 1 when the output threshold a_2 is increased. However, the transition is abrupt when the input threshold a_0 is decreased. The neuronal spikes around the activation sites in a Class 0 neuronal CA decays, either fast or slow, across the patch. This characteristic is analogous by the inhibitory neurons in that the neurons tend to inhibit its neighbors in synchrony. In Class 1 neuronal CA, spike trains are self-sustained across the patch over the simulation duration, a behavior exhibited by excitatory neurons.

Acknowledgment

Reinier Xander A. Ramos acknowledges the Department of Science and Technology (DOST) for his Advanced Science and Technology Human Resources Development Program (ASTHRDP) scholarship.

References

[1] L. Lapicque, Recherches quantitatives sur l'excitation électrique des nerfs traitée comme une polarisation, *J. Physiol. Pathol. Gén.* **9**, 620 (1907).

[2] A. L. Hodgkin and A. F. Huxley, A quantitative description of membrane current and its application to conduction and excitation in nerve, *J. Physiol.* **117**, 500 (1952).

[3] W. Gerstner, W. M. Kistler, R. Naud, and L. Paninski, *Neuronal Dynamics: From single neurons to networks and models of cognition and beyond* (Cambridge University Press, UK, 2014), https://neuronaldynamics.epfl.ch/online/index.html.

[4] J. C. S. Pang and J. Y. Bantang, Hodgkin-Huxley neurons with defective and blocked ion channels, *Int. J. Mod. Phys. C* **26**, 1550112 (2015).

[5] R. Ramos and J. Bantang, Proposed cellular automaton model for a neuronal patch with a thresholded linear activation function, in *Proceedings of the 36th Samahang Pisika ng Pilipinas Physics Conference* (2018), vol. 36, SPP-2018-PB-37.

[6] S. Wolfram, *A New Kind of Science* (Wolfram Media, Inc., Champaign, Illinois, 2002), https://www.wolframscience.com/nks/.

[7] K. A. Hawick and C. J. Scogings, Cycles, transients, and complexity in the game of death spatial automaton, in *Proc. International Conference on Scientific Computing (CSC'11)*, edited by H. R. Arabnia and G. A. Gravvanis (CSREA Press, 2011), 241–247.

[8] H. Lodish, A. Berk, S. L. Zipursky, et al., The Action Potential and Conduction of Electric Impulses, in *Molecular Cell Biology* (W. H. Freeman, New York, 2000), 4th ed.

[9] M. A. Patestas and L. P. Gartner, *A Textbook of Neuroanatomy* (John Wiley & Sons, 2016).

Verhulst and bifurcation analyses of a neuronal network on an outer-totalistic toroidal cellular automata

Reinier Xander A. Ramos* and Johnrob Y. Bantang

National Institute of Physics, University of the Philippines Diliman, Quezon City, Philippines
*Corresponding author: rramos@nip.upd.edu.ph

Abstract

Our cellular automata (CA) model of a neuronal patch is a fast way of exploring the dynamics of population of interconnected neurons. In this study, we analyzed the cobweb diagrams of our model and found that the nonzero steady-state neuronal activity is an emergent property of our model. We investigated if there exists a bifurcation and found that the period-doubling happens at negatively-sloped activation function. Neurons exhibiting this characteristic implies a possible way of modelling epileptic neurons. We propose the categorization of neuronal CA as follows: Class 0 (Zero steady-state), Class 1 (Nonzero steady-state), and Class 2 (Periodic steady-state).

Keywords: cellular automata, neuron, activation function

1 Introduction

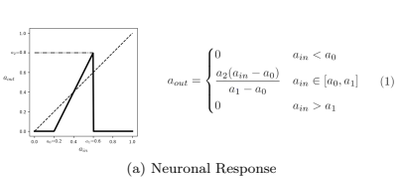
Computational neuronal models have been improved since Louis Lapicque developed the integrate-and-fire model of a single neuron [1] in 1907. However, research studies about the dynamics of a network of neurons (i.e. a neuronal patch) are challenging because of the huge amount of computational power needed to simulate millions or billions of neurons.

In our previous work [2, 3], we showed that it is possible to model a neuronal patch using simple rules implemented on a 100 x 100 cellular automaton. We studied the dynamics of this neuronal patch given various lattices and boundary conditions as shown in Figure 1 [4, 5]. The rule is simple: each neuron in the cellular automaton (CA) follows the same activation function described in Figure 2a.

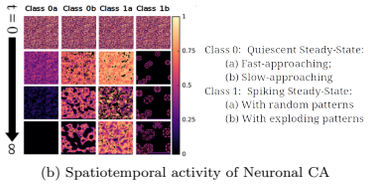
The parameters a_0, a_1, and a_2 are thresholds set so that the activation function is a first-order approximation of the response function of a Hodgkin-Huxley neuron [1, 6]. The parameter thresholds were varied from 0 to 1 (normalized value) in steps of 0.1. We have shown that our dynamical categories

(a) Totalistic Moore neighborhood (b) Outer-totalistic Moore neighborhood (c) Toroidal Lattice (d) Spherical Lattice

Figure 1: Types of neighborhood and lattices that were used in the simulation of neuronal cellular automaton.



$$a_{out} = \begin{cases} 0 & a_{in} < a_0 \\ \dfrac{a_2(a_{in} - a_0)}{a_1 - a_0} & a_{in} \in [a_0, a_1] \\ 0 & a_{in} > a_1 \end{cases} \quad (1)$$



(a) Neuronal Response (b) Spatiotemporal activity of Neuronal CA

Figure 2: Left: Simplified neuronal response used in our CA model derived from the response function of a Hodgkin-Huxley neuron. The output a_{out} is related to the probability of the neuron to fire given the input a_{in} by Equation 1. Right: Spatiotemporal activity of each Neuronal CA Class. The activity are obtained by varying the parameter thresholds a_0, a_1, a_2.

were invariant to various 2D and quasi-3D lattices, neighborhood, and boundary conditions. The resulting spatiotemporal activity was then categorized into two classes as shown in Figure 2b. These classification were consistent to the resulting neuronal types of Hodgkin-Huxley [1, 2].

In this paper, we investigated qualitatively the dynamics of a neuron in our cellular automata via Verhulst or cobweb diagrams. We compared our observation to the over-all behavior of the CA as described from the state-space trajectories. Furthermore, we investigated the bifurcation diagrams of our neuronal CA including a downward-sloped activation and found the conditions at which epileptic neurons activate.

2 Analytical Diagrams

State-space trajectories are obtained by plotting a_{t+1} vs a_t, with the $a = 0$ (quiescent) located at the bottom left of the plot, and $a = 1$ (highest activity) located at the top right of the plot. The trajectory lets us know if there are attractors on the dynamical behavior.

In Verhulst diagrams (or cobweb diagrams), we can visualize qualitatively how a dynamical system behaves over time [7]. Given an initial condition, we trace our iterated function, Equation 1 shown in Figure 2a against the line $y = x$. This is done so that the next input state is the previous output state. If the trajectory is an inward spiral, then the neuron exhibits stable steady-state. If it is outwards spiral, the neuron exhibits an unstable steady-state. If a square-like trajectory is achieved, then the neuron is in an oscillating steady-state.

Bifurcation diagrams show how the behavior of a dynamical system change as we vary a parameter of interest [8]. We examined the presence of period doubling as we vary one of the parameter thresholds while keeping the other two constant. From our previous simulations, we have shown that a duration of 100 timesteps is enough for the simulation to reach steady-state at any given parameter set [2, 3]. We obtained the average neuronal patch activity $\langle a \rangle$ for the last ten timesteps. The values were plotted against varying parameter thresholds.

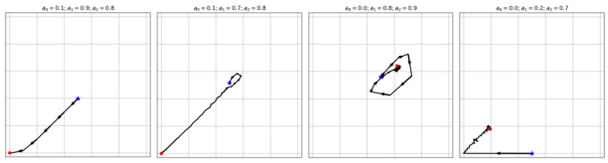


Figure 3: Representative state-space trajectories for each of the neuronal CA classes: (from left to right) Class 0a, Class 0b, Class 1a, Class 1b. The $\rightarrow$ marks the path or trajectory of the steady-state neuronal activity.

3 Numerical Results

In Figure 3, the steady-state trajectory for each neuronal CA class is shown. The trajectory describes the behavior of the CA as if it is a singular neuronal patch. The lower left region indicates the lowest activity (quiescent state) of the patch while the upper right region indicates the highest activity (maximum spiking state). It is evident that two leftmost trajectories exhibit quiescent steady-state while the other two have spiking steady-state. Thus, our classification in Figure 2b holds.

We can compare these trajectories with each of the possible trajectories of a single neuron shown in the Verhulst diagrams in Figure 4, given different initial conditions that the neuron may assume at the zeroth timestep. For Class 0a, the activation function is below the $y = x$. Hence, any neuron will go towards quiescent state regardless of the initial state (marked by the blue star $\star$). Collectively, these neurons will approach quiescent state in a short amount of timesteps. In Class 0b, the activation function crosses the line $y = x$ once. Any neuron that started to the left or to the right of intersection point maintains an activity but reaches quiescent steady-state. A neuron initialized at exactly at the intersection point, however, stays in that value, but only a few of these are present in our CA since we initialized at a uniform random distribution. Neurons in Class 0b collectively goes to quiescent state but a slower rate relative to Class 0a because of the presence of the intersection point. Inhibitory neurons can be modelled by Class 0 neuronal patch. For Class 1 CA, the intersection point is located at $y = x = 0$. The collection of neurons will approach a nonzero steady-state. The greater the difference between a_0 and a_1, the longer the CA will reach steady-state. Neuronal CA with shorter steady-state time produces exploding patterns (Class

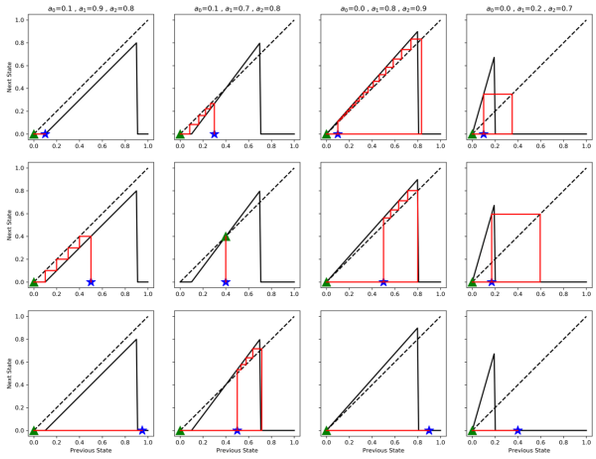


Figure 4: Each column shows the Verhulst diagrams for each of the neuronal classes: (from left to right) Class 0a, Class 0b, Class 1a, Class 1b. Each row shows different possible cases for initial condition (marked by ⋆) for that neuronal class. The ▲ marks the final state on x-axis.

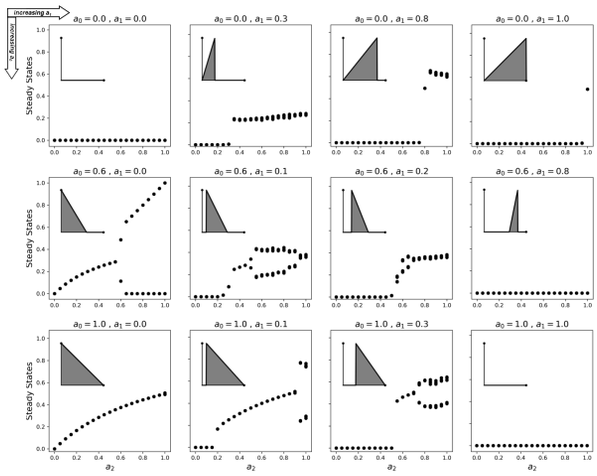


Figure 5: Representative bifurcation diagrams for our neuronal CA. Each plot shows the steady-state values vs increasing a_2. From left column to right shows increasing a_1 while from top row to bottom shows increasing a_0. The inset shows the activation function for that parameter set given that $a_2 = 1.0$.

1b) as opposed to the random patterns (Class 1a) on longer steady-state. These patterns are shown in Figure 2b. Excitatory neurons belong to Class 1 in this scheme.

For the bifurcation diagrams, we performed an exhaustive investigation and plotted representatives in Figure 5. Individual plot shows the steady-state values against varying a_2. As we increase a_1 (from left to right), the period-doubling happens at a higher a_2 values. As we increase a_0 (from top to bottom), the

steady-state values also increase. The cases where $\{a_0 = 0.0, a_1 = 0.0\}$ and $\{a_0 = 1.0$ and $a_1 = 1.0\}$ are trivial. Furthermore, period-doubling of the steady-state happens only when the activation function is negatively-sloped and strictly satisfies $a_1 = 0$. We investigated further this case and found out that only at these specific constraints will the neuron oscillate as shown in the Verhulst diagrams in Figure 6. Here, each neuron in the CA has 75% chance of oscillating (indicated by the square-like trajectory), and these individual trajectories tend to synchronize resulting in a periodic oscillation shown in the state-space trajectory (Figure 6, rightmost). Epileptic neurons are known to exhibit periodic oscillating activity [9] and thus can be modelled by our negatively-sloped activation function.

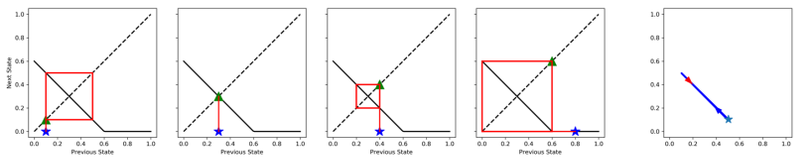


Figure 6: The four leftmost plots show the cobweb diagrams for each initial state (marked by $\star$) for an epileptic neuronal CA with threshold values $a_0 = 0.6$, $a_1 = 0.0$, and $a_2 = 0.6$. The ▲ marks the final state. The rightmost plot shows the state-space trajectory of the steady-state activity. The $\star$ marks $t = 0$. The $\rightarrow$ shows the trajectory and the $\rightarrow$ indicates that the trajectory travels back to the starting point.

4 Conclusions

Using our simplified neuronal CA, we modelled three different types of neurons: excitatory, inhibitory, and epileptic. The cobweb analyses showed that for Class 1, individual neurons initialized at different states will mostly reach a quiescent steady-state. However, when these neurons are interconnected, the steady-state of the patch will be spiking. The patterns of the spiking activity is dependent on the difference between a_0 and a_1. The bifurcation diagrams revealed that a period-doubling occurs when the activation function is negatively-sloped and satisfies $a_1 = 0$. The cobweb diagrams for this case verified a periodic steady-state which resembles epileptic neuronal behavior.

References

[1] W. Gerstner, W. M. Kistler, R. Naud, and L. Paninski, *Neuronal Dynamics: From single neurons to networks and models of cognition and beyond* (Cambridge University Press, Cambridge, England, 2014), https://neuronaldynamics.epfl.ch/online/index.html.

[2] R. Ramos and J. Bantang, Proposed cellular automaton model for a neuronal patch with a thresholded linear activation function, in *Proceedings of the Samahang Pisika ng Pilipinas* (2018), vol. 36, SPP-2018-PB-37.

[3] R. X. Ramos and J. Bantang, Totalistic cellular automata model of a neuronal network on a spherical surface, in *Proceedings of the Samahang Pisika ng Pilipinas* (2019), vol. 37, SPP-2019-PB-16.

[4] R. Ramos and J. Bantang, Classification of the dynamics of an outer-totalistic 2d and quasi-3d cellular automata simplistic models of neuronal patches, in *16th International Conference on Molecular Systems Biology*, BST 2019 (De La Salle University, Manila, 2019), 51, poster presentation, https://drive.google.com/file/d/1bs655p68uv0SpZmmOcDr8MFV-_bWLvo_/view.

[5] R. Ramos and J. Bantang, An outer-totalistic 2d and quasi-3d cellular automata simplistic models of neuronal patches, in *Brain Connects 2019 and the 9th Neuroscience International Symposium*, St. Luke's Medical Center (Global City, Taguig, 2019), BC2019-14, poster and oral presentation.

[6] A. L. Hodgkin and A. F. Huxley, A quantitative description of membrane current and its application to conduction and excitation in nerve, *J. Physiol.* **117**, 500 (1952).

[7] Wolfram Mathworld, Web diagram, http://mathworld.wolfram.com/WebDiagram.html, accessed: 2020-02-07.

[8] C. Çelik Karaaslanlı, in *Numerical Simulation: From Theory to Industry*, edited by M. Andriychuk (INTECH Open Access Publisher, Pidstryhach Institute for Applied Problems of Mechanics and Mathematics, Ukraine, 2012), chap. 1, 3-34, http://doi.org/10.5772/50075.

[9] R. A. Stefanescu, S. R. G, and T. S. S, Computational models of epilepsy, *Seizure* **21**, 748 (2012).

Totalistic cellular automata model of a neuronal network on a spherical surface

Reinier Xander A. Ramos* and Johnrob Y. Bantang

National Institute of Physics, University of the Philippines Diliman
*Corresponding author: rramos@nip.upd.edu.ph

Abstract

Our understanding of how the brain works is still an open challenge. Current neuronal models (e.g. integrate-and-fire, Hodgkin-Huxley) are able to mimic voltage patterns in neurons using ordinary differential equations. Coupling these models would be difficult to solve manually and numerically. Previously, we proposed a cellular automata neuronal model that would efficiently simulate the dynamics of a large number of neurons. We chose a linear activation function that mimics the neuronal response of an integrate-and-fire neuron. In this paper, we extended our analysis by comparing nontotalistic vs totalistic modes. We also investigated how a spherical lattice topology affects the neuronal network dynamics as opposed to the previous toroidal topology. Finally, we were able to find three dynamical categories that we previously observed, by considering the length of the activation function and the fraction of initially active neurons.

Keywords: Neuronal network dynamics (87.19.lj), Structures and organization in complex systems (89.75.Fb), Models of single neurons and networks (87.19.ll)

1 Introduction

All activities of multicellular organisms are processed in the brain. From walking, finding food and eating, watching TED-Ed videos, and even when sleeping—the brain is always active. The brain processes these activities through membrane potential difference that move across the connected neurons through the gaps in-between neurons called synapses. How these signals flow inside a neuron is analytically described by current neuronal models such as integrate-and-fire (IAF) model [1] and Hodgkin-Huxley (HH) model [2]. However, these models use ordinary differential equations to describe a single neuron. This imposes a problem when modelling a neuronal network as it will require coupling of many differential equations. It will be inefficient to solve this problem both manually and numerically.

In this paper, we aim to improve our previous work [3] on a cellular automata (CA) neuronal model that simulates a biological neuronal patch. Each neuron behaves similar to an IAF or an HH neuron. The model consists of 10000 neuronal patch arranged in a 100 x 100 lattice, with a nontotalistic toroidal Moore neighborhood (8 neighbors). The rules of the cellular automaton follows the response function given by Equation 1:

$$a_{out} = a_2 \frac{a_{in} - a_0}{a_1 - a_0}, \qquad a_0 \leq a_{in} \leq a_1 \tag{1}$$

and $a_{out} = 0$, otherwise. The parameters a_0, a_1, and a_2 are the thresholds for the response function shown in Figure 1. Here, the parameter a represents the net activity of the neuron. In the spatiotemporal regime, the variations in the net activity were observed to behave similar to a signal that propagates across automaton. The input parameter a_{in} of a neuron was obtained from the average a-value of its neighbors, and the output parameter a_{out} determines the next state of the neuron.

We concluded our previous work with a categorization observed by varying each of these three threshold parameters from 0 to 1 (with an increment of 0.1). There were three possible types of CA: a) Category-0, fast-approaching zero steady-state; b) Category-1, slowly-approaching zero steady-state; and c) Category-2, nonzero steady-state. The neuronal response of a representative CA for each category were compared to that of the HH neuronal response with varying input current strength.

We extend our analysis to a totalistic CA wherein the neighborhood of a cell includes itself. This property is exhibited by the IAF and HH models, where the membrane potential of the neuron depends on its current value, as well as the difference in ion concentration on its the surroundings. We also considered a spherical topology, detailed further in Section 2. This topology captures the three-dimensional nature of the neocortical neurons found on the surface of the brain. We characterized our CA model by obtaining

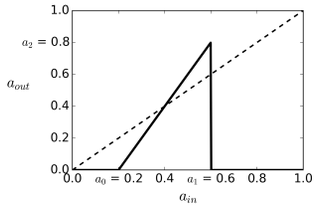


Figure 1: Sample linear activation function bounded by the thresholds a_0, a_1, and a_2. The dotted line represent $a_{out} = a_{in}$, which we used to as a reference to characterize our cellular automata.

the length of activation to predict at what specific quantifiers will we observe a CA that falls on each category. This characterization test is further discussed in Section 3.

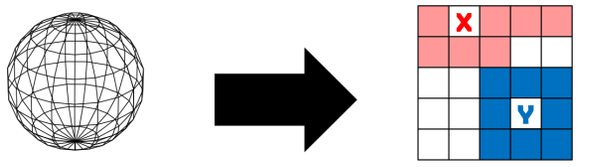
Totalistic vs Nontotalistic CA. In a nontotalistic CA, the input a_{in} takes only the average of the neighbors of a cell. On the other hand, in a totalistic CA, the a-value of the cell is included in calculating the average. We analyzed both the nontotalistic and totalistic CA for both toroidal and spherical topology.

$$\begin{aligned} a_{in,nontotalistic} &= \frac{\sum a_{neighbors}}{\#ofneighbors} \\ a_{in,totalistic} &= \frac{a_{self} + \sum a_{neighbors}}{1 + \#ofneighbors} \end{aligned} \tag{2}$$

2 Toroidal vs spherical topology

We constructed a network of 10000 neurons in two different topology. For the toroidal topology, the neurons were placed in a 100 x 100 grid, with each cell representing a neuron. The cells in the top layer of the grid were connected to the bottom layer, and the cells at the leftmost part of the grid were connected to the cells in the rightmost part, creating a wrap-around effect on the grid.

For the spherical topology, we considered the sphere shown in Figure 2a. Notice that the cells at the pole are fully connected to each other because they share the same boundary at the pole itself. However, the cells in the equator still have the regular Moore neighborhood. Thus, if we flatten out the surface of the sphere, we will obtain a lattice similar to that in Figure 2b, only that the top and bottom layers are fully connected with each other. However, we still have to connect cells in the leftmost part with the cells on the rightmost part to obtain the closed surface of the sphere.



(a) Spherical surface wireframe

(b) Flattened spherical surface

Figure 2: Construction of the spherical topology by flattening the spherical surface. In addition to the three neighbors at the second row, cell X is also fully connected to all the cells in the top row (red-filled cells). On the other hand, cell Y on the equator still has the regular Moore neighborhood (blue-filled cells).

3 Update rules

We initialized the grid by assigning each cell a random a-value obtained from a uniform distribution from 0 to 1. This a-value represents the net activity of the neuron. For each time-step we took the average a-values of the neighbors as the input a_{in} to that cell. The response function in Equation 1was then

used to obtain the next state a_{out} of the cell. The update was done simultaneously to all cells up to 200 time-steps.

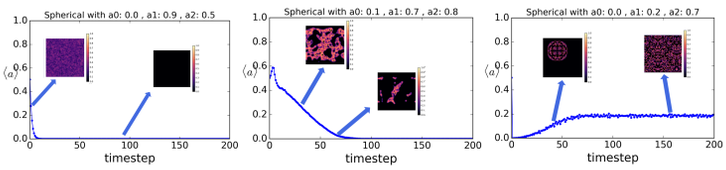
We categorized the resulting dynamics by the following method: 1) if the system reached a zero steady-state at $t \leq 10$, then it falls as Category-0; 2) if the system reached a zero steady-state at $10 < t \leq 200$, then the CA belongs to Category-1; and 3) if the system reached a nonzero steady-state at $t = 200$, then it is a Category-2 automaton. These time values were set by comparing the $\langle a \rangle$ over time, for each CA. Furthermore, the cellular automata within each category exhibits similar spatiotemporal behavior.

To find out at what particular thresholds a_0, a_1, and a_2, will result to which category of CA, we counted the fraction of neurons that are activated at $t = 0$. These were the neurons that have initial a-value in the range of $[a_{intersection}, a_1]$, where $a_{intersection}$ is the intersection point of response function and the line $a_{out} = a_{in}$. This fraction were the only neurons that will fire before the next time-step. We plotted this fraction versus the length of the activation function according to the following.

$$d_{act} = \sqrt{a_2^2 + (a_1 - a_0)^2} \tag{3}$$

4 Numerical Experiments

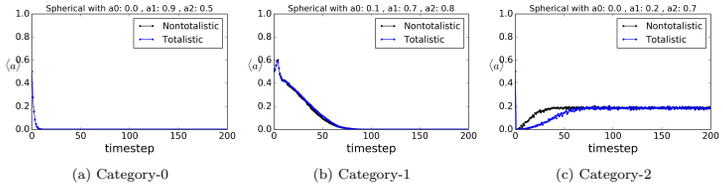
Figure 3 compares the average response of a totalistic CA using toroidal and spherical topology. We found that spherical topology has generally lower average activation than the toroidal CA. This was observed for all categories but was easily noticeable for Category-1. The lower average response was due to the signal being concentrated on top and bottom of the sphere (polar cells), rather than freely flowing across the whole lattice in toroidal topology. All of these observations were also seen for nontotalistic CA.



(a) Category-0 (b) Category-1 (c) Category-2

Figure 3: Average response of the totalistic CA with spherical topology. Inset: snapshot of the CA at the time-step specified by the arrow.

The average response $\langle a \rangle$ of the automata over time between the nontotalistic and totalistic CA for spherical topology shows that there was a delay in steady-state activation when totalistic rule was considered, as shown in Figure 4c and 4b. The delay shows that by taking into account the current state of the neuron, the neuron inhibits itself to fire, unless the a-value of the neuron is overcome by the total a-values of the neighbors. However, the delay was barely noticeable in Category-0 due to the exponential decay of $\langle a \rangle$. This behavior was also observed for all categories in toroidal topology. In Hodgkin-Huxley and integrate-and-fire models [1, 2], the current voltage state of the neuron is considered in updating to its next state. This phenomenon was also observed in a biological neuron [4]. Hence, totalistic rules are more biologically consistent than the nontotalistic rules.



(a) Category-0 (b) Category-1 (c) Category-2

Figure 4: Average response of the spherical CA comparing totalistic and nontotalistic rules. Note that the totalistic CA has a delay in reaching the steady-state as compared with nontotalistic.

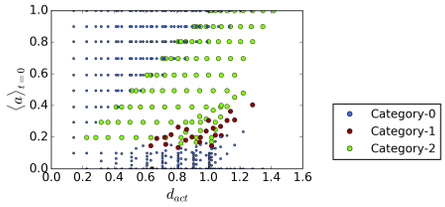


Figure 5: Length of activation versus fraction of initially activated neurons for totalistic spherical CA. The upper-left region and the region below $\langle a \rangle_{t=0} = 0.1$ are assured to be fast-approaching steady-state CA. The lower-right region just above $\langle a \rangle_{t=0} = 0.1$ is where Category-1 CA would be observed. The central region consists of the nonzero steady-state CA.

Figure 5 shows a plot of the length of activation d_{act} vs the fraction of initially activated neurons for a totalistic spherical CA. We found that at least 20% of the neurons at $t = 0$ must be active or spiking to observe a nonzero steady-state CA (Category-2). This happens if the length of activation is between 0.2 and 0.6. Furthermore, a region from the center to the upper-right corner of the plot ensures a Category-2 automaton (green circles). In the lower-right region of the plot, we see that it is possible to observe either Category-2 or Category-1 automaton (red circles). Hence, this region exhibits a mixed-state property. Thus, by setting the specific thresholds, the CA will either result to a nonzero steady-state or a CA that slowly saturates to a zero steady-state. This property is observed on bistable or gating neurons [5]. These neurons switch their states between hyperpolarized (zero) state or depolarized (nonzero) state [6]. The same behavior was observed for a totalistic toroidal CA.

5 Conclusions

As a summary, we found that the average response of a CA with spherical boundary had the same behavior as with the toroidal boundary. However the average values was lower for spherical because the activity is being concentrated on the polar rows. This behavior characterizes the neocortical neurons which are found on the surface of the brain. Furthermore, we found that a totalistic CA takes longer time-step to reach steady-state than the nontotalistic rules that we used in our previous work. However, the totalistic CA captures the biological nature of the neuron as presented in the Hodgkin-Huxley model. Furthermore, we found that there were observable regions for each of our categories by looking at the d_{act} vs $\langle a \rangle_{t=0}$. A definite region of purely nonzero steady-state can be observed. However, there are regions that inhibit mixed-state property. This region can be attributed to gating neurons, which switch between depolarized (nonzero) or hyperpolarized (zero) state.

References

[1] D. Tal and E. Schwartz, Computing with the leaky integrate-and-fire neuron: logarithmic computation and multiplication, *Neural Comput.* **9**, 305 (1997).

[2] A. L. Hodgkin and A. F. Huxley, A quantitative description of membrane current and its application to conduction and excitation in nerve, *J. Physiol.* **117**, 500 (1952).

[3] R. Ramos and J. Bantang, in *Proceedings of the 36th Samahang Pisika ng Pilipinas* (2018), SPP–2018–PB–37.

[4] J. C. Dalton and R. FitzHugh, Applicability of Hodgkin-Huxley model to experimental data from the giant axon of lobster, *Science* **131**, 1533 (1960).

[5] AnimatLab, Bistable neuron, http://animatlab.com/Help/Documentation/Neural-Network-Editor/Neural-Simulation-Plug-ins/Firing-Rate-Neural-Plug-in/Bistable-Neuron.

[6] C. E. Stafstrom, To depolarize or hyperpolarize? at the axon initial segment, E_{GABA} sets the stage, *EPI Epilepsy Currents* **9**, 28 (2009), pMID: 19396347.

Proposed cellular automaton model for a neuronal patch with a thresholded linear activation function

Reinier Xander A. Ramos* and Johnrob Y. Bantang

National Institute of Physics, University of the Philippines Diliman
*Corresponding author: rramos@nip.upd.edu.ph

Abstract

Hodgkin-Huxley, and most of our current neuronal models consist of differential equations to explain the behavior of a single neuron with an electrical synapse, a chemical synapse, or both. In order to model a network of neurons, we need to couple these differential equations, and consequently, increasing the number of equations to solve. This will be difficult to do manually, and even, subjected to the limits of computing power of todays computers, when done numerically. In this paper, we propose a model for a network of neurons (N = 10 000) placed in a 100 x 100 lattice, with a linear activation function as the rule of the cellular automaton.

Keywords: Neuronal network dynamics, Models of single neurons and networks, Structures and organization in complex systems

1 Introduction

Many of the neuronal models that are widely used today are composed of differential equations to illustrate the dynamics of a neuron. Most of these, which involves only one neuron, includes the Hodgkin-Huxley model [1] and integrate-and-fire model [2]. While these are good representations of a neuron, as verified by comparing to experimental data [3, 4], it would be difficult to model a network of neurons using only differential equations. The differential equations will be coupled to signify a network of neurons. However, the number of differential equations to be solve increases with the number of neurons of the network that we want to analyze. It will be tedious to solve this manually and will be subjected to the digital limitations when solved analytically. This imposes a problem when dealing with a large network of neurons – specifically, the brain, since the human brain has about 10^{10} - 10^{11} neurons, each with 10^4 - 10^5 connections or synapses.

Furthermore in these models, the response of a neuron is represented by the firing rate as a function of time. This can be interpreted as the activation function of the neuron, and it has the same form for most of the current neuronal models. To be able to analyze a network of neurons, we propose a cellular automaton model with a linear activation function, as show in Figure 1) parametrized by a_0, a_1, and a_2, as the rule for the automaton. This linear activation function is the simplest but similar way to describe how a neuron fires over time. We further note that this activation function does not directly correspond to the actual neuronal response over time, but refers to the probability of firing at that specific time.

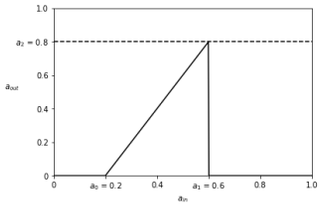


Figure 1: Sample linear activation function bounded by the thresholds a_0, a_1, and a_2.

2 Methodology

In this paper, we considered a 100 x 100 lattice, where each node of the lattice represents a neuron. Each node has 8 neighbors (via Moore neighborhood, r = 1). We initialize the grid by assigning uniformly random a (the parameter a stands for the net activity, which is analogous to the probability of firing

at that specific time-step) values for each cell. Then we implement an activation function, which is constructed by the combination of a Heaviside and a linear function defined by Equation 1, as the generation rule of the automaton. All the parameters vary from 0 to 1, with an increment of 0.1. The same initial configuration of a-values were used for different parameter set. The update for each cell is as follows: 1) we take the average a-values of the eight (8) neighbors, 2) this average will be the input parameter a_{in}, and 3) the next state of the cell a_{out} is given by the activation function defined as follows:

$$a_{out} = a_2 \frac{a_{in} - a_0}{a_1 - a_0}, \qquad a_0 \leq a_{in} \leq a_1 \tag{1}$$

and $a_{out} = 0$, otherwise.

We obtained an animation for each test, while taking the average a of the whole network as a function of time (1 time-step = 1 generation). To get the actual neuronal patch response, we define three discrete states (Q = quiescent or inactive; F = firing or spiking; R = refractory period). These three states were inspired by the same states in the Brian's brain automaton [5]. We initialized the patch to be in inactive state. For each update, if the previous state is Q, then it has a probability $\langle a \rangle$ to fire (F). From state F, it will always go to state R, and stays R for the next two (2) time-steps. Then, from state R, it will always go back to Q. These are summarized in Figure 2.

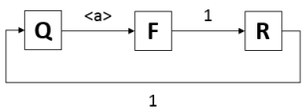


Figure 2: The state rules applied to the patch for each update. Q represents quiescent or inactive state; F stands for firing state; and R stand for the refractory period. The arrow labels correspond to the probability of changing the state at the next time-step.

3 Results and Discussion

We found that 100 generations or updates are enough to observe steady-state of the network. There were three kinds of transient activity found before going to a steady-state. Category 1 is a neuronal network that becomes inactive very fast, and is exhibited by short linear activation function (a_1-$a_0 < 0.2$, for $a_0 < 0.5$). This is because the input-output mapping is very limited to small range that most of the cells tend to be inactive at the next update. This type of steady-state is also exhibited by all tests with $0.3 \leq a_0$. This might be due to the small range of possible steady-state values that is not zero. Examples of these are shown in Figure 3.

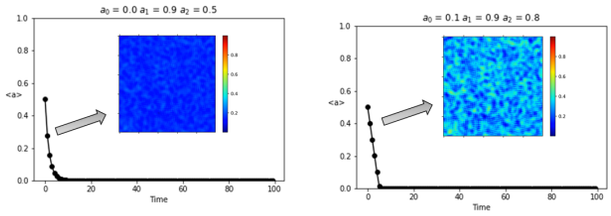


Figure 3: Average a-values as a function of time for Category 1 Neuronal patch. (Inset: snapshot of the patch at the time-step specified by the arrow).

Category 2 shows a steady-state above zero. There are two subtypes for this kind. One subtype produces a "firework"-patterns after a few seconds, then evolves into a chaotic behavior. This subtype happens with the parameters set to: 1) $a_0 = 0$, $a_1 = 0.2$, and $0.3 \leq a_2 \leq 1$; and 2) $a_0 = 0.1$, $a_1 = 0.3$, and $0.8 \leq a_2 \leq 1$. The other subtype also gives a nonzero steady-state value but produces chaotic behavior

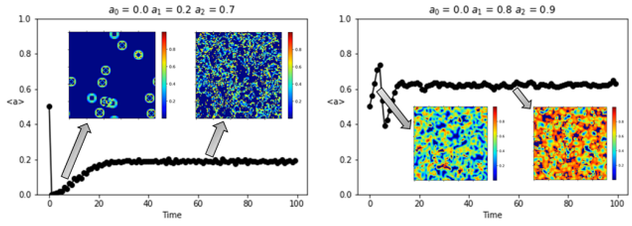


Figure 4: Average a-values as a function of time for Category 2 Neuronal patch. (Inset: snapshot of the patch at the time-step specified by the arrow).The figure in the left also shows a snapshot of the "firework" pattern and snapshot of the steady-state; as compared to the other subtype of Category 2, shown on the right.

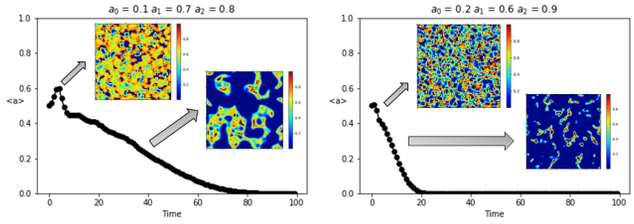


Figure 5: Average a-values as a function of time for Category 3 Neuronal patch. (Inset: snapshot of the patch at the time-step specified by the arrow). Notice that the most of the cells become inactive(blue) in such a short period.

immediately and does not produce any pattern. This happens mostly when $a_0 = 0$, or 0.1. Together these two subtypes, after a long time, gives an average value of below 0.5. We also observe an abrupt drop of the average a after a short time.This is why those with cut-off of $a_0 \geq 0.5$ are included in Category 1. One of each of these subtypes are shown for comparison in Figure 4.

The last one, Category 3 shows a neuronal network with slowly decreasing average a-value, until it reaches a zero steady-state. This is exhibited by an activation mapping with large range of possible input-output mapping, and with low a_0 values. Two samples are shown in Figure 5.

Now, we apply the state rules from Figure 2 to find the neuronal patch response over time for each category. The results are shown in Figure 6. For Category 1, only a few (usually one) spikes was observed. This is because the probability of firing goes and stays zero immediately. The most number of spikes were observed for Category 2 (middle), due to the nonzero steady-state of $\langle a \rangle$. For Category 3, we observe a number of spikes at the beginning, but then will vanish after a long period of time-steps as we expect according to the firing probability over time as seen in Figure 5. These neuronal patch responses resemble the spiking behavior of a single Hodgkin-Huxley neuron [1]. Also, these plots were found to be similar in behavior with the results of Luccioli et al [6].

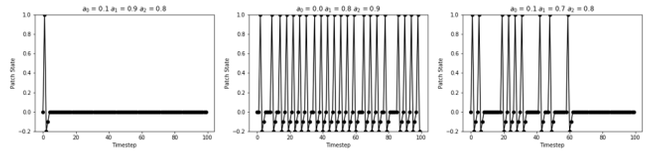


Figure 6: Neuronal patch responses for Category 1 (left), Category 2 (center), and Category 3 (right).

4 Conclusions

We conclude that it is possible to model a patch of neurons using a cellular automaton by the implementation of a thresholded linear activation. We found that we can categorize the different parameter tests into three: 1)fast-approaching to inactive patch; 2) a patch with a steady-state $\langle a \rangle$; and 3) slow but approaching to an inactive patch. We also obtained the response of the neuronal patch over time by introducing the state rules presented, and we obtain similar behavior with the response of a single Hodgkin-Huxley neuron.

References

[1] A. L. Hodgkin and A. F. Huxley, A quantitative description of membrane current and its application to conduction and excitation in nerve, *J. Physiol.* **117**, 500 (1952).

[2] D. Tal and E. Schwartz, Computing with the leaky integrate-and-fire neuron: logarithmic computation and multiplication, *Neural Comput.* **9**, 305 (1997).

[3] J. C. Dalton and R. FitzHugh, Applicability of Hodgkin-Huxley model to experimental data from the giant axon of lobster, *Science* **131**, 1533 (1960).

[4] H. Motz and F. Rattay, A study of the application of the Hodgkin-Huxley and the Frankenhaeuser-Huxley model for electrostimulation of the acoustic nerve, *Neuroscience* **18**, 699 (1986).

[5] M. S. Evans, Cellular Automata - Brians Brain (2002), last accessed 18 May 2018, `http://www.msevans.com/automata/briansbrain.html`.

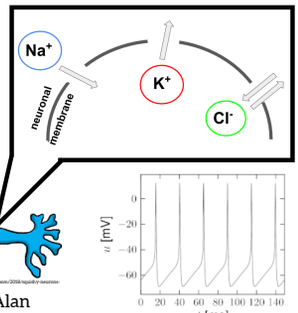
[6] S. Luccioli, T. Kreuz, and A. Torcini, Dynamical response of the Hodgkin-Huxley model in the high-input regime, arXiv:cond-mat/0512341 (2006).

An outer-totalistic 2D and quasi-3D cellular automata simplistic models of neuronal patches

Reinier Xander A. Ramos* and Johnrob Y. Bantang
National Institute of Physics, University of the Philippines, Diliman, Quezon City
*rramos@nip.upd.edu.ph; jybantang@gmail.com

BC2019-14

Hodgkin-Huxley (HH) Model of a Neuron



- Developed in 1952 by Alan Hodgkin and Andrew Huxley
- Describes how action potential across neuronal membranes propagates through difference in ion concentration

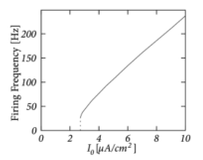
$$I = C_m \frac{dV_m}{dt} + \bar{g}_K n^4 (V_m - V_K) + \bar{g}_{Na} m^3 h (V_m - V_{Na}) + \bar{g}_l (V_m - V_l)$$

$$\frac{dn}{dt} = \alpha_n(V_m)(1-n) - \beta_n(V_m)n$$

$$\frac{dm}{dt} = \alpha_m(V_m)(1-m) - \beta_m(V_m)m$$

$$\frac{dh}{dt} = \alpha_h(V_m)(1-h) - \beta_h(V_m)h$$

HH Neuronal Response [1, 2]

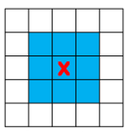


HH Activation Function [1, 2]

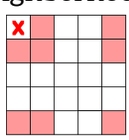
Cellular Automata (CA) Model

- A neuron is represented by its activity ***a*** with value ranging from 0 to 1 (arbitrary units).
- 10^4 neurons arranged in a 100 x 100 lattice

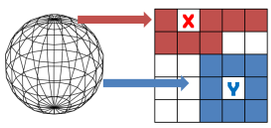
Neighborhood and Lattices



Outer-totalistic Moore neighborhood



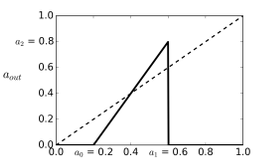
Toroidal lattice



Spherical Lattice

Rule: Activation Function

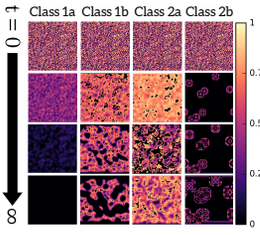
- Linear function derived from HH activation function
- Input signal a_{in} taken from the average of the neighbors
- Output signal a_{out}

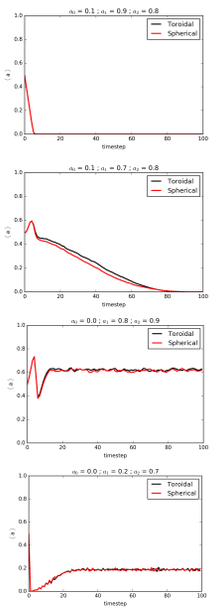


$$a_{out} = \begin{cases} a_2 \frac{a_{in} - a_0}{a_1 - a_0}, & a_0 \le a_{in} \le a_1 \\ 0, & otherwise \end{cases}$$

Classification of Neuronal Patch Activity

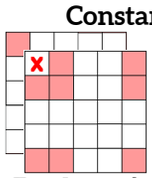
1. Resting steady-state
   a. Quickly approaching inactive state
   b. Slowly approaching inactive state
2. Spiking steady-state
   a. Random patterns produced
   b. Exploding patterns produced



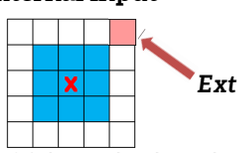


Neuronal patch activity of each for each of the CA classes. [4, 5]

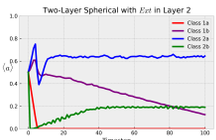
Two-Layered Lattice and Addition of Constant External Input



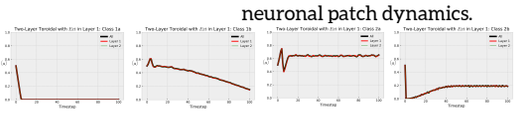
Two-Layered Toroidal Lattice



A single neuron always have a value of one (***a=1***) for that cell.



- The neuronal activity for each classes are maintained with the inclusion of ***Ext***.
- Much more can be explored using the proposed simplistic model that adequately mimics neuronal patch dynamics.



References

1. A. L. Hodgkin and A. F. Huxley. A quantitative description of membrane current and its application to conduction and excitation in nerve. J. Physiol., 117(54):500–544, 1952.
2. W. Gerstner, W. M. Kistler, R. Naud, and L. Paninski, Neuronal Dynamics: From single neurons to networks and models of cognition (Cambridge University Press, 2014), 2nd ed., https://neuronaldynamics.epfl.ch/online/index.html.
3. A New Kind of Science | Wolfram Media Inc., Champaign, Illinois, US. United States, 2002). ISBN1-57955-008-8.
4. R. Ramos and J. Bantang. Proposed cellular automaton model for a neuronal patch with a thresholded linear activation function, Proceedings of the Samahang Pisika ng Pilipinas 36, SPP-2018-PB-37 (2018). URL: https://paperview.spp-online.org/proceedings/article/view/SPP-2018-PB-37.
5. R. X. Ramos and J. Bantang. Totalistic cellular automata model of a neuronal network on a spherical surface, Proceedings of the Samahang Pisika ng Pilipinas 37, SPP-2019-PB-16 (2019). URL: https://paperview.spp-online.org/proceedings/article/view/SPP-2019-PB-16.

Classification of the dynamics of an outer-totalistic 2D and quasi-3D cellular automata simplistic models of neuronal patches

Reinier Xander A. Ramos* and Johnrob Y. Bantang

Complexity Science Group, Instrumentation Physics Laboratory, National Institute of Physics, National Science Complex, University of the Philippines, Diliman, Quezon City

*rramos@nip.upd.edu.ph; jybantang@nip.upd.edu.ph

Hodgkin-Huxley (HH) Model of a Neuron

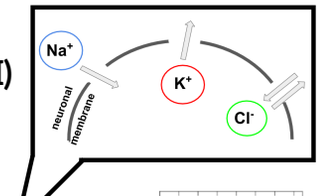


Image taken from https://www.brainfacts.org/for-educators/for-the-classroom/2018/squishy-neurons-activity-031918

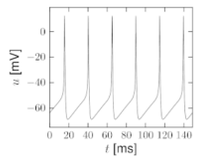
- Developed in 1952 by Alan Hodgkin and Andrew Huxley
- Describes how action potential across neuronal membranes propagates through difference in ion concentration

$$I = C_m\frac{dV_m}{dt} + \bar{g}_K n^4(V_m - V_K) + \bar{g}_{Na} m^3 h(V_m - V_{Na}) + \bar{g}_l(V_m - V_l)$$

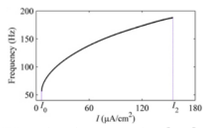
$$\frac{dn}{dt} = \alpha_n(V_m)(1-n) - \beta_n(V_m)n$$

$$\frac{dm}{dt} = \alpha_m(V_m)(1-m) - \beta_m(V_m)m$$

$$\frac{dh}{dt} = \alpha_h(V_m)(1-h) - \beta_h(V_m)h$$



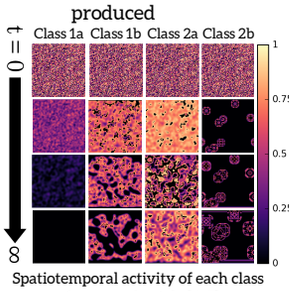
HH Neuronal Response [1, 2]



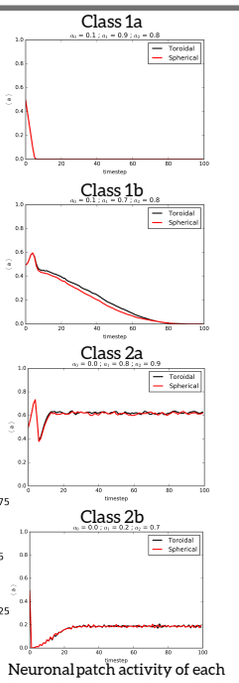
HH Activation Function [1, 2]

Classification of Neuronal Patch Activity

1. Resting steady-state
   a. Quickly approaching inactive state
   b. Slowly approaching inactive state
2. Spiking steady-state
   a. Random patterns produced
   b. Exploding patterns produced



Spatiotemporal activity of each class of neuronal patch. [4, 5]

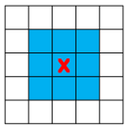


Neuronal patch activity of each for each of the CA classes. [4, 5]

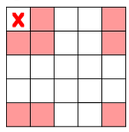
Cellular Automata (CA) Model

- A neuron is represented by its activity ***a*** with value ranging from 0 to 1 (arbitrary units).
- 10^4 neurons arranged in a 100 x 100 lattice

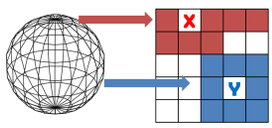
Neighborhood and Lattices



Outer-totalistic Moore neighborhood



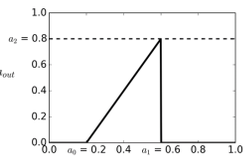
Toroidal lattice



Spherical Lattice

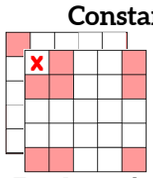
Rule: Activation Function

- Linear function derived from HH activation function
- Input signal a_{in} taken from the average of the neighbors
- Output signal a_{out}

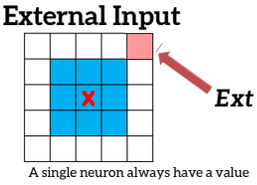


$$a_{out} = \begin{cases} a_2\frac{a_{in}-a_0}{a_1-a_0}, & a_0 \le a_{in} \le a_1 \\ 0, & otherwise \end{cases}$$

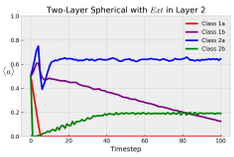
Two-Layered Lattice and Addition of Constant External Input



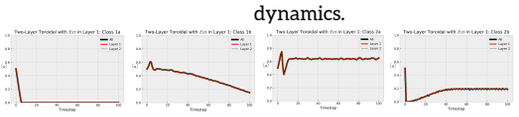
Two-Layered Toroidal Lattice



A single neuron always have a value of one (***a=1***) for that cell.



- The neuronal activity for each classes are maintained with the inclusion of ***Ext***.
- Much more can be explored using the proposed simplistic model that adequately mimics neuronal patch dynamics.



References

1. A. L. Hodgkin and A. F. Huxley. A quantitative description of membrane current and its application to conduction and excitation in nerve. J. Physiol., 117(54)500–544, 1952.
2. W. Gerstner, W. M. Kistler, R. Naud, and L. Paninski, Neuronal Dynamics: From single neurons to networks and models of cognition (Cambridge University Press, 2014), 2nd ed., https://neuronaldynamics.epfl.ch/online/index.html.
3. A New Kind of Science (Wolfram Media Inc., Champaign, Illinois, US, United States, 2002), ISBN1-57955-008-8.
4. R. Ramos and J. Bantang. Proposed cellular automaton model for a neuronal patch with a thresholded linear activation function, Proceedings of the Samahang Pisika ng Pilipinas 36, SPP-2018-PB-37 (2018). URL: https://paperview.spp-online.org/proceedings/article/view/SPP-2018-PB-37.
5. R. X. Ramos and J. Bantang. Totalistic cellular automata model of a neuronal network on a spherical surface, Proceedings of the Samahang Pisika ng Pilipinas 37, SPP-2019-PB-16 (2019). URL: https://paperview.spp-online.org/proceedings/article/view/SPP-2019-PB-16.

Appendix B

Machine System Specifications

CPU	AMD Ryzen 7 3750H (Octa-core, 2.3GHz)
RAM	16384 MB
OS	Windows 11 Home Single Language (10.0, Build 22621)
Arch	64-Bit

Table 1: Machine and device specifications used for the simulations.

Python	3.9.16
IPython	7.31.1
Anaconda	23.3.1
Spyder	5.3.3
numpy	1.23.5
matplotlib	3.6.2
Pillow	9.4.0
numba	0.56.4
tqdm	4.65.0

Table 2: Module and package versions used for the simulations.

Appendix C

Simulation Code

```
# -*- coding: utf-8 -*-
"""
Created on Thu Jan 19 18:11:16 2023

@author: ramos
"""

import os
import itertools as it
from tqdm import tqdm
import numpy as np
from numpy import random as rnd
import numba as nb
import glob
from PIL import Image
from matplotlib import pyplot as plt
from matplotlib import colors as mplc
import matplotlib as mpl

def main(make_directory=False, make_base=False, make_external=False,
         make_layered=False, make_global=False, make_local=False):
    automatonType           = 'base'
    if make_directory:      makeDirectory()

    parametersList = np.load(f'./parameters-list/parameters-{parametersLength}-{functionType}.npy')
    pathPrefix     = f'./lattice-{latticeSize}/case-{parametersLength}/{functionType}'

    if make_base:
        make_spatiotemporal_directory(pathPrefix, parametersList,automatonType)
```

```
30          generateContSptm_raw(pathPrefix, parametersList, ↩
                automatonType)
31          plotContSptm_png(pathPrefix, parametersList, ↩
                automatonType)
32          plotContSptm_gif(pathPrefix, parametersList, ↩
                automatonType)
33          generateContMean_raw(pathPrefix, parametersList, ↩
                automatonType)
34          plotContMean_png(pathPrefix, parametersList, ↩
                automatonType)
35
36          generateDiscSptm_raw(pathPrefix, parametersList, ↩
                automatonType)
37          plotDiscSptm_png(pathPrefix, parametersList, ↩
                automatonType)
38          plotDiscSptm_gif(pathPrefix, parametersList, ↩
                automatonType)
39          generateDiscMean_raw(pathPrefix, parametersList, ↩
                automatonType)
40          plotDiscMean_png(pathPrefix, parametersList, ↩
                automatonType)
41
42      if make_external:
43          fullExternalList = np.load(f'./lattice-{latticeSize}/↩
                external-input-\
44                                      index-list.npy')
45          for fractionExternal in fractionExternalList:
46              externalIndicesList = fullExternalList[:int(↩
                    fractionExternal*numberNeurons)]
47              automatonType       = f'external-{fractionExternal}↩
                    '
48              make_spatiotemporal_directory(pathPrefix, ↩
                    parametersList, automatonType)
49              generateContSptm_raw(pathPrefix, parametersList, ↩
                    automatonType, externalIndicesList=↩
                    externalIndicesList)
50              plotContSptm_png(pathPrefix, parametersList, ↩
                    automatonType, fractionExternal=↩
                    fractionExternal)
51              plotContSptm_gif(pathPrefix, parametersList, ↩
                    automatonType, fractionExternal=↩
                    fractionExternal)
52              generateContMean_raw(pathPrefix, parametersList, ↩
                    automatonType, fractionExternal=↩
                    fractionExternal)
53              plotContMean_png(pathPrefix, parametersList, ↩
                    automatonType, fractionExternal=↩
                    fractionExternal)
54
```

```
55              generateDiscSptm_raw(pathPrefix, parametersList, ↩
                    automatonType)
56              plotDiscSptm_png(pathPrefix, parametersList, ↩
                    automatonType, fractionExternal=↩
                    fractionExternal)
57              plotDiscSptm_gif(pathPrefix, parametersList, ↩
                    automatonType, fractionExternal=↩
                    fractionExternal)
58              generateDiscMean_raw(pathPrefix, parametersList, ↩
                    automatonType, fractionExternal=↩
                    fractionExternal)
59              plotDiscMean_png(pathPrefix, parametersList, ↩
                    automatonType, fractionExternal=↩
                    fractionExternal)
60
61      if make_layered:
62          for numberLayers in numberLayersList:
63              automatonType       = f'layered-{numberLayers}'
64              make_spatiotemporal_directory(pathPrefix, ↩
                    parametersList, automatonType)
65              generateContSptm_raw(pathPrefix, parametersList, ↩
                    automatonType, numberLayers=numberLayers)
66              plotContSptm_png(pathPrefix, parametersList, ↩
                    automatonType, numberLayers=numberLayers)
67              plotContSptm_gif(pathPrefix, parametersList, ↩
                    automatonType, numberLayers=numberLayers)
68              generateContMean_raw(pathPrefix, parametersList, ↩
                    automatonType, numberLayers=numberLayers)
69              plotContMean_png(pathPrefix, parametersList, ↩
                    automatonType, numberLayers=numberLayers)
70
71              generateDiscSptm_raw(pathPrefix, parametersList, ↩
                    automatonType, numberLayers=numberLayers)
72              plotDiscSptm_png(pathPrefix, parametersList, ↩
                    automatonType, numberLayers=numberLayers)
73              plotDiscSptm_gif(pathPrefix, parametersList, ↩
                    automatonType, numberLayers=numberLayers)
74              generateDiscMean_raw(pathPrefix, parametersList, ↩
                    automatonType, numberLayers=numberLayers)
75              plotDiscMean_png(pathPrefix, parametersList, ↩
                    automatonType, numberLayers=numberLayers)
76
77      if make_global:
78          make_global_directory(pathPrefix, parametersList, ↩
                automatonType)
79          generateGlobalPhaseSpace_raw(pathPrefix, parametersList↩
                , automatonType, lastTimesteps_PS_BD)
80          plotGlobalPhaseSpace_png(pathPrefix, parametersList, ↩
                automatonType)
```

```
81          generateGlobalBifurcation_raw(pathPrefix, ↩
                parametersList, automatonType, lastTimesteps_PS_BD)
82          plotGlobalBifurcation_png(pathPrefix, parametersList, ↩
                automatonType)
83      if make_local:
84          make_local_directory(pathPrefix, parametersList)
85          generateCobwebPoints_raw(pathPrefix, parametersList)
86          plotCobwebDiagram_png(pathPrefix, parametersList)
87          generateFixedPoints_raw(pathPrefix, parametersList, ↩
                lastTimesteps_PS_BD)
88          plotFixedPoints_png(pathPrefix, parametersList)
89          generateSSPhaseSpace_raw(pathPrefix, parametersList, ↩
                lastTimesteps_PS_BD)
90          plotSSPhaseSpace_png(pathPrefix, parametersList)
91
92      endMessage = 'END'
93      endMessage = endMessage.center(len(endMessage) + 4, ' ')
94      print(f'\n{endMessage:=^{len(endMessage)+50}}')
95
96  #%% Dynamic Parameters
97  latticeChoice           = 0
98  parametersChoice        = 1
99  functionChoice          = 1
100 nbcChoice               = 1
101
102 numberTrials            = 5
103 DURATION                = 100
104 neuronIDx, neuronIDy    = 12, 0   # must not be from external ↩
        indices
105
106 axislabelsFontsize     = 17
107 titleFontsize          = 17
108 ticklabelsFontsize     = 15
109 dpiSize                = 300
110 colorbarlabelsFontsize = 15
111
112 #%% Static Parameters
113 latticeSizeList         = [100, 1024]
114 parametersCaseList      = [6, 11, 21]
115 functionTypeList        = ['linear', 'nonlinear']
116 analysisTypeList        = ['spatiotemporal', 'global', 'local',↩
        'spike-statistics']
117 nbcList                 = ['toroidal-outer', 'toroidal-total', ↩
        'spherical-outer', 'spherical-total']
118 spatiotemporalList      = ['cont-sptm', 'cont-mean', 'disc-sptm↩
        ', 'disc-mean']
119 subpathList             = ['raw', 'png', 'gif']
120 fractionExternalList    = [0.01, 0.05, 0.1]
121 numberLayersList        = [2, 3, 4]
```

```
122
123 parameterSymbolList      = [['a_0', 'a_1', 'a_2'], ['a_0', 'a_2'↩
    , 'b']]
124 parameterDirectoryList  = [['a0' , 'a1' , 'a2' ], ['a0' , 'a2' ↩
    , 'b']]
125 parameterNameList        = [['Min Input Threshold', 'Max Input ↩
    Threshold', 'Max Output Threshold'],
126                           ['Input Threshold', 'Output ↩
                                Threshold', 'Nonlinearity']]
127 phaseSpaceIndex          = list(it.combinations([0,1,2], r=2))
128 phaseSpaceCaseList       = [f'{_p}-vs-{_q}' for [_p, _q] in it.↩
    combinations(parameterDirectoryList[functionChoice], r=2)]
129 bifurcationCaseList      = [f'vary-{_p}' for _p in ↩
    parameterDirectoryList[functionChoice]]
130 localCaseList            = ['cobweb-diagram', 'fixed-point-↩
    bifurcation',
131                           'phase-space-steady-state', 'phase-↩
                                space-period']
132 statisticsList           = ['probability-distribution', 'shannon↩
    -entropy', 'thermo-entropy', 'spike-train',
133                           'population-density']
134
135 cobwebTimesteps          = 100
136 cobwebPrecision          = 10
137
138 colorList                = ['k', 'r', 'y', 'g', 'b', 'pink']
139 discreteResponseList     = [-0.2, 0, 1]
140 activationFunctionsList = [activateLinear, activateNonlinear]
141 activationFunction       = activationFunctionsList[↩
    functionChoice]
142 nbcBaseFunctionsList     = [get_toroidal_outer_neighbors, ↩
    get_toroidal_total_neighbors,
143                           get_spherical_outer_neighbors, ↩
                                get_spherical_total_neighbors]
144 nbcLayeredFunctionsList = [get_toroidal_outer_neighbors_layered↩
    , get_toroidal_total_neighbors_layered,
145                           get_spherical_outer_neighbors_layered↩
                                , ↩
                                get_spherical_total_neighbors_layered↩
                                ]
146
147 latticeSize              = latticeSizeList[latticeChoice]
148 parametersLength         = parametersCaseList[parametersChoice]
149 functionType             = functionTypeList[functionChoice]
150
151 gridCoordinates          = np.asarray(list(it.product(range(↩
    latticeSize), repeat = 2)))
152 timeList                 = range(DURATION+1)
153 L                        = latticeSize
```

```
154 numberNeurons            = latticeSize * latticeSize
155 R,Q,F                    = range(3)
156
157 lastTimesteps_PS_BD       = 10
158 phaseSpaceCoordinates     = list(it.product(range(↩
    parametersLength), repeat=2))
159 parameterNames            = parameterNameList[functionChoice]
160
161 entropyPDbins             = round(np.sqrt(L*L))
162 PDweights                 = np.ones(L*L)/(L*L)
163
164 sptcmap                   = plt.get_cmap('magma')
165 RQFcmap                   = mplc.ListedColormap(['rebeccapurple'↩
    , 'black','khaki'])
166 figureParameters          = {  'axes.labelsize' : ↩
    axislabelsFontsize
167                             , 'axes.titlesize' : titleFontsize
168                             , 'xtick.labelsize': ↩
                                  ticklabelsFontsize
169                             , 'ytick.labelsize': ↩
                                  ticklabelsFontsize
170                             , 'savefig.dpi'    : dpiSize
171                             , 'image.origin'   : 'lower'
172                            }
173 mpl.rcParams.update(figureParameters)
174
175 if parametersChoice == 1 or (parametersChoice, functionChoice) ↩
    == (0,0):
176     round_ = 1
177 elif parametersChoice == 2 or (parametersChoice, functionChoice↩
    ) == (0,1):
178     round_ = 2
179 referenceList           = np.round(np.linspace(0, 1, ↩
    parametersLength), round_)
180
181 if parametersChoice in [1,2]:
182     nbcList                 = [nbcList[nbcChoice]]
183     nbcBaseFunctionsList    = [nbcBaseFunctionsList[nbcChoice]]
184     nbcLayeredFunctionsList = [nbcLayeredFunctionsList[↩
        nbcChoice]]
185
186 #%% Initialization
187 def makeDirectory():
188     for lattice, case, function, analysis in it.product(↩
        latticeSizeList,parametersCaseList,functionTypeList,↩
        analysisTypeList):
189         os.makedirs(f'./lattice_{lattice}/case_{case}/{function↩
            }/{analysis}', exist_ok=True)
190     generateParameters()
```

```
191     generateInitialStates(numberTrials)
192     generateNeuronFiringProbability(latticeSize)
193     generateIndicesExternalInput(latticeSize)
194     generateSptmColorbars()
195 
196 def generateSptmColorbars():
197     os.makedirs(path:='./spatiotemporal-colorbars', exist_ok=↩
            True)
198     plt.ioff()
199     fig, ax = plt.subplots(subplot_kw=dict(aspect='equal', ↩
            xticks=[], xticklabels=[], yticks=[], yticklabels=[]))
200     im      = ax.imshow(np.zeros((latticeSize,latticeSize), ↩
            dtype=np.float32), cmap='magma', vmin=0, vmax=1)
201     cbar    = fig.colorbar(im, ax=ax, pad=0.02)
202     cbar.set_ticks(ticks=[0, 0.5, 1], labels=['0.0', '0.5', '↩
            1.0'], fontsize=ticklabelsFontsize)
203     cbar.set_label('Neuronal Activity, $a$', rotation = 270, ↩
            labelpad = 15, fontsize = colorbarlabelsFontsize)
204     ax.remove()
205     plt.savefig(f'{path}/cont-sptm.png', bbox_inches='tight')
206 
207     os.makedirs(path:='./spatiotemporal-colorbars', exist_ok=↩
            True)
208     plt.ioff()
209     fig, ax = plt.subplots(subplot_kw=dict(aspect='equal', ↩
            xticks=[], xticklabels=[], yticks=[], yticklabels=[]))
210     im      = ax.imshow(np.zeros((latticeSize,latticeSize), ↩
            dtype=np.float32), cmap='magma', vmin=0, vmax=1)
211     cbar    = fig.colorbar(im, ax=ax, pad=0.02)
212     cbar.set_ticks(ticks=[0, 0.5, 1], labels=['0.0', '0.5', '↩
            1.0'], fontsize=ticklabelsFontsize)
213     cbar.set_label(r'Average steady-state, $\langle a \rangle_↩
            {\infty}$', rotation = 270, labelpad = 15, fontsize = ↩
            colorbarlabelsFontsize)
214     ax.remove()
215     plt.savefig(f'{path}/cont-mean.png', bbox_inches='tight')
216 
217     fig, ax = plt.subplots(subplot_kw=dict(aspect='equal', ↩
            xticks=[], xticklabels=[], yticks=[], yticklabels=[]))
218     im = ax.imshow(np.zeros((L,L), dtype=np.float32), cmap=↩
            RQFcmap, vmin=R, vmax=F)
219     cbar = fig.colorbar(im, ax=ax, pad=0.02)
220     cbar.set_ticks(ticks=[0, 1, 2], labels=['R', 'Q', 'F'], ↩
            fontsize=ticklabelsFontsize)
221     cbar.set_label('Neuronal Discrete State', rotation = 270, ↩
            labelpad = 15, fontsize = colorbarlabelsFontsize)
222     ax.remove()
223     plt.savefig(f'{path}/disc-sptm.png', bbox_inches='tight')
224     plt.close('all')
```

```
225
226 @nb.njit()
227 def populateGrid(latticeSize):
228     return rnd.random(size=(latticeSize,latticeSize)).astype(np.float32)
229
230 def generateInitialStates(numberTrials):
231     os.makedirs(f'./lattice-{latticeSize}/initial-states', exist_ok=True)
232     LOG_initial_pos = LOGFILE.tell()
233     for trial in tqdm(range(numberTrials), desc='Generating Initial CA States'):
234         LOGFILE.seek(LOG_initial_pos)
235         initialGrid = populateGrid(latticeSize)
236         np.save(f'./lattice-{latticeSize}/initial-states/{trial}.npy', initialGrid)
237
238 def generateNeuronFiringProbability(latticeSize):
239     os.makedirs(f'./lattice-{latticeSize}/neuron-firing-probability', exist_ok=True)
240     for t in timeList:
241         grid = populateGrid(latticeSize)
242         np.save(f'./lattice-{latticeSize}/neuron-firing-probability/{t}.npy', grid)
243
244 def generateIndicesExternalInput(latticeSize):
245     rnd.shuffle(externalGrid := gridCoordinates.copy())
246     np.save(f'./lattice-{latticeSize}/external-input-index-list.npy', externalGrid)
247
248 def generateParameters():
249     os.makedirs('./parameters-list', exist_ok=True)
250     for case, function in it.product(parametersCaseList, functionTypeList):
251         if   case == 11 or (case, function) == (6, 'linear'): _round = 1
252         elif case == 21 or (case, function) == (6, 'nonlinear'):   _round = 2
253         aList = np.round(np.linspace(0, 1, case), _round)
254         if case > 10 and function=='linear':
255             parametersList = np.asarray(list(it.product(aList, repeat=3)))
256         elif case < 10 and function=='linear':
257             parametersList = np.round(np.asarray([[0.1, 0.9, 0.8]       # Class 0a
258                                                   ,[0.1, 0.7, 0.8]      # Class 0b
```

```
259                                                     ,[0.0, 0.8, ↩
                                                           0.9] ↩
                                                                  # ↩
                                                           Class 1a
260                                                     ,[0.0, 0.2, ↩
                                                           0.7] ↩
                                                                  # ↩
                                                           Class 1b
261                                                     ,[0.6, 0.0, ↩
                                                           0.6] ↩
                                                                  # ↩
                                                           Class 2
262                                                     ,[0.8, 0.0, ↩
                                                           0.9] ↩
                                                                  # ↩
                                                           Class 2
263                                                    ]), _round)
264         elif case == 11 and function=='nonlinear':
265             bList = np.round(np.linspace(0, 5, case), _round)
266             parametersList = np.asarray(list(it.product(aList, ↩
                    aList, bList)))
267         elif case == 21 and function=='nonlinear':
268             bList = np.linspace(0, 40, case)
269             parametersList = np.asarray(list(it.product(aList, ↩
                    aList, bList)))
270         elif case < 10 and function=='nonlinear':
271             parametersList = np.round(np.asarray([[0.45, 0.38, ↩
                    1.5]     # young
272                                                     ,[0.29, 1.00, ↩
                                                           2.2]       #↩
                                                            aged
273                                                     ,[0.1, 0.7, ↩
                                                           2.0] ↩
                                                                  # ↩
                                                           Class 0a
274                                                     ,[0.2, 1.0, ↩
                                                           1.5] ↩
                                                                  # ↩
                                                           Class 0b
275                                                     ,[0.0, 0.7, ↩
                                                           1.5] ↩
                                                                  # ↩
                                                           Class 1a
276                                                     ,[0.0, 0.9, ↩
                                                           2.0] ↩
                                                                  # ↩
                                                           Class 1b
277                                                    ]), _round)
```

```
278         np.save(f'./parameters-list/parameters-{case}-{function}', parametersList)
279
280 def make_spatiotemporal_directory(pathPrefix, parametersList, automatonType):
281     for nbc, sptm, subpath, [p1,p2,p3] in tqdm(it.product(nbcList, spatiotemporalList, subpathList, parametersList), desc='Making spatiotemporal directory'):
282         os.makedirs(f'{pathPrefix}/spatiotemporal/{nbc}/{automatonType}/{sptm}', exist_ok=True)
283         if ('mean' in sptm and subpath in ['raw', 'png']) or 'sptm' in sptm:
284             os.makedirs(f'{pathPrefix}/spatiotemporal/{nbc}/{automatonType}/{sptm}/{subpath}', exist_ok=True)
285         if ('sptm' in sptm and subpath in ['raw', 'png']) or ('layered' in automatonType and 'mean' in sptm and subpath=='raw'):
286             os.makedirs(f'{pathPrefix}/spatiotemporal/{nbc}/{automatonType}/{sptm}/{subpath}/{p1}-{p2}-{p3}', exist_ok=True)
287
288 def make_global_directory(pathPrefix, parametersList, automatonType):
289     for nbc, psCase, subpath in tqdm(it.product(nbcList, phaseSpaceCaseList, subpathList[:2]), desc='Making global phase-space directory'):
290         os.makedirs(f'{pathPrefix}/global/{nbc}/{automatonType}/phase-space/{psCase}/{subpath}', exist_ok=True)
291
292     for nbc, bdCase, subpath in tqdm(it.product(nbcList, bifurcationCaseList, subpathList[:2]), desc='Making global bifurcation-diagram directory'):
293         os.makedirs(f'{pathPrefix}/global/{nbc}/{automatonType}/bifurcation/{bdCase}/{subpath}', exist_ok=True)
294
295 def make_local_directory(pathPrefix, parametersList):
296     for localCase in tqdm(localCaseList, desc='Making local directory'):
297         if localCase == 'cobweb-diagram':
298             for subpath, [p1,p2,p3] in it.product(subpathList[:2], parametersList):
299                 os.makedirs(f'{pathPrefix}/local/{localCase}/{subpath}/{p1}-{p2}-{p3}', exist_ok=True)
300         elif localCase == 'fixed-point-bifurcation':
301             for subpath in subpathList[:2]:
302                 os.makedirs(f'{pathPrefix}/local/{localCase}/{subpath}', exist_ok=True)
```

```
303         else:
304             for subpath, bifCase in it.product(subpathList[:2],↩
                    bifurcationCaseList):
305                 os.makedirs(f'{pathPrefix}/local/{localCase}/{↩
                        subpath}/{bifCase}', exist_ok=True)
306
307 #%% Methods
308
309 #%%% Activation Functions
310 @nb.njit(nb.float32(nb.float32, nb.float64, nb.float64, nb.↩
        float64))
311 def activateLinear(a_in, a0, a1, a2):
312     # if a0 == a1:
313     #     return 0.
314     if a0 == a1 and a_in == a0:
315         return a2
316     elif a0 == a1 and a_in != a0:
317         return 0.
318     elif min(a0,a1) <= a_in <= max(a0,a1):
319         return a2*(a_in-a0)/(a1-a0)
320     else:
321         return 0.
322
323 @nb.njit(nb.float32(nb.float32, nb.float64, nb.float64, nb.↩
        float64))
324 def activateNonlinear(a_in, a0, a2, b):
325     # if 1-a0 == 0 or a_in == 1:
326     #     return 0
327     if a0 == 1 and a_in == a0:
328         return a2
329     elif a0 == 1 and a_in != a0:
330         return 0.
331     elif a_in >= a0:
332         return a2 * (1-np.exp((-b) * (-np.log(1-(a_in-a0)/(1-a0↩
                )))) )
333     else:
334         return 0
335
336 #%%% Base Neighborhood and Boundary Conditions
337 @nb.njit(nb.float32(nb.int32, nb.float32[:,:], nb.int32, nb.↩
        int32))
338 def get_toroidal_outer_neighbors(L, grid, j, i):
339     neighbors = np.asarray([grid[j-1, i-1],   grid[j, i-1],   ↩
            grid[j-L+1, i-1],
340                             grid[j-1, i],                     ↩
                                grid[j-L+1, i],
341                             grid[j-1, i-L+1], grid[j, i-L+1], ↩
                                grid[j-L+1, i-L+1]])
342     return np.mean(neighbors)
```

```
343
344 @nb.njit(nb.float32(nb.int32, nb.float32[:,:], nb.int32, nb.int32))
345 def get_toroidal_total_neighbors(L, grid, j, i):
346     neighbors = np.asarray([grid[j-1, i-1],   grid[j, i-1],   grid[j-L+1, i-1],
347                             grid[j-1, i],     grid[j, i],     grid[j-L+1, i],
348                             grid[j-1, i-L+1], grid[j, i-L+1], grid[j-L+1, i-L+1]])
349     return np.mean(neighbors)
350
351 @nb.njit(nb.float32(nb.int32, nb.float32[:,:], nb.int32, nb.int32))
352 def get_spherical_outer_neighbors(L, grid, j, i):
353     if j == 0:
354         neighbors = np.empty(L-1+3).astype(np.float32)
355         neighbors[:i] = grid[j,:i]
356         neighbors[i:L-1] = grid[j,i+1:L]
357         neighbors[L-1:] = [grid[j+1,i-1], grid[j+1,i], grid[j+1,i-L+1]]
358     elif j == L-1:
359         neighbors = np.empty(L-1+3).astype(np.float32)
360         neighbors[:i] = grid[j,:i]
361         neighbors[i:L-1] = grid[j,i+1:L]
362         neighbors[L-1:] = [grid[j-1,i-1], grid[j-1,i], grid[j-1,i-L+1]]
363     else:
364         neighbors = np.array([grid[j-1, i-1  ], grid[j, i-1  ], grid[j-L+1, i-1  ],
365                               grid[j-1, i    ], grid[j-L+1, i    ],
366                               grid[j-1, i-L+1], grid[j, i-L+1], grid[j-L+1, i-L+1]])
367     return np.mean(neighbors)
368
369 @nb.njit(nb.float32(nb.int32, nb.float32[:,:], nb.int32, nb.int32))
370 def get_spherical_total_neighbors(L, grid, j, i):
371     if j == 0:
372         neighbors = np.empty(L+3).astype(np.float32)
373         neighbors[:L] = grid[j,:]
374         neighbors[L:] = [grid[j+1,i-1], grid[j+1,i], grid[j+1,i-L+1]]
375     elif j == L-1:
376         neighbors = np.empty(L+3).astype(np.float32)
377         neighbors[:L] = grid[j,:]
```

```
378         neighbors[L:] = [grid[j-1,i-1], grid[j-1,i], grid[j-1,i-L+1]]
379     else:
380         neighbors = np.array([grid[j-1, i-1  ], grid[j, i-1  ], grid[j-L+1, i-1  ],
381                               grid[j-1, i    ], grid[j, i    ], grid[j-L+1, i    ],
382                               grid[j-1, i-L+1], grid[j, i-L+1], grid[j-L+1, i-L+1]])
383     return np.mean(neighbors)
384
385 #%% Layered Neighborhood and Boundary Conditions
386 @nb.njit(nb.float32(nb.int32, nb.float32[:,:,:], nb.int32, nb.int32, nb.int32, nb.int32))
387 def get_toroidal_outer_neighbors_layered(L, grid, j, i, numLayers, layer):
388     neighbors = np.asarray([grid[layer, j-1, i-1],   grid[layer, j, i-1],   grid[layer, j-L+1, i-1],
389                             grid[layer, j-1, i], grid[layer, j-L+1, i],
390                             grid[layer, j-1, i-L+1], grid[layer, j, i-L+1], grid[layer, j-L+1, i-L+1]])
391     if numLayers == 2:
392         layerNeighbors = np.asarray([grid[layer-1, j, i]])
393     else:
394         layerNeighbors = np.asarray([grid[layer-numLayers+1, j, i], grid[layer-1, j, i]])
395     allNeighbors = np.empty(len(neighbors)+len(layerNeighbors)).astype(np.float32)
396     allNeighbors[:len(neighbors)] = neighbors
397     allNeighbors[len(neighbors):] = layerNeighbors
398     return np.mean(allNeighbors)
399
400 @nb.njit(nb.float32(nb.int32, nb.float32[:,:,:], nb.int32, nb.int32, nb.int32, nb.int32))
401 def get_toroidal_total_neighbors_layered(L, grid, j, i, numLayers, layer):
402     neighbors = np.asarray([grid[layer, j-1, i-1],   grid[layer, j, i-1],   grid[layer, j-L+1, i-1],
403                             grid[layer, j-1, i],     grid[layer, j, i],     grid[layer, j-L+1, i],
404                             grid[layer, j-1, i-L+1], grid[layer, j, i-L+1], grid[layer, j-L+1, i-L+1]])
405     if numLayers == 2:
406         layerNeighbors = np.asarray([grid[layer-1, j, i]])
```

```
407     else:
408         layerNeighbors = np.asarray([grid[layer-numLayers+1, j,↩
                i], grid[layer-1, j, i]])
409     allNeighbors = np.empty(len(neighbors)+len(layerNeighbors))↩
            .astype(np.float32)
410     allNeighbors[:len(neighbors)] = neighbors
411     allNeighbors[len(neighbors):] = layerNeighbors
412     return np.mean(allNeighbors)
413 
414 @nb.njit(nb.float32(nb.int32, nb.float32[:,:,:], nb.int32, nb.↩
        int32, nb.int32, nb.int32))
415 def get_spherical_outer_neighbors_layered(L, grid, j, i, ↩
        numLayers, layer):
416     if j == 0:
417         neighbors = np.empty(L-1+3).astype(np.float32)
418         neighbors[:i] = grid[layer, j,:i]
419         neighbors[i:L-1] = grid[layer, j,i+1:L]
420         neighbors[L-1:] = [grid[layer, j+1,i-1], grid[layer, j↩
                +1,i], grid[layer, j+1,i-L+1]]
421     elif j == L-1:
422         neighbors = np.empty(L-1+3).astype(np.float32)
423         neighbors[:i] = grid[layer, j,:i]
424         neighbors[i:L-1] = grid[layer, j,i+1:L]
425         neighbors[L-1:] = [grid[layer, j-1,i-1], grid[layer, j↩
                -1,i], grid[layer, j-1,i-L+1]]
426     else:
427         neighbors = np.array([grid[layer, j-1, i-1  ], grid[↩
                layer, j, i-1  ], grid[layer, j-L+1, i-1  ],
428                               grid[layer, j-1, i    ], ↩
                                                    grid[↩
                                   layer, j-L+1, i    ],
429                               grid[layer, j-1, i-L+1], grid[↩
                                   layer, j, i-L+1], grid[layer,↩
                                    j-L+1, i-L+1]])
430     if numLayers == 2:
431         layerNeighbors = np.asarray([grid[layer-1, j, i]])
432     else:
433         layerNeighbors = np.asarray([grid[layer-numLayers+1, j,↩
                i], grid[layer-1, j, i]])
434     allNeighbors = np.empty(len(neighbors)+len(layerNeighbors))↩
            .astype(np.float32)
435     allNeighbors[:len(neighbors)] = neighbors
436     allNeighbors[len(neighbors):] = layerNeighbors
437     return np.mean(allNeighbors)
438 
439 @nb.njit(nb.float32(nb.int32, nb.float32[:,:,:], nb.int32, nb.↩
        int32, nb.int32, nb.int32))
440 def get_spherical_total_neighbors_layered(L, grid, j, i, ↩
        numLayers, layer):
```

```
441     if j == 0:
442         neighbors = np.empty(L+3).astype(np.float32)
443         neighbors[:L] = grid[layer, j,:]
444         neighbors[L:] = [grid[layer, j+1,i-1], grid[layer, j+1,↩
                i], grid[layer, j+1,i-L+1]]
445     elif j == L-1:
446         neighbors = np.empty(L+3).astype(np.float32)
447         neighbors[:L] = grid[layer, j,:]
448         neighbors[L:] = [grid[layer, j-1,i-1], grid[layer, j-1,↩
                i], grid[layer, j-1,i-L+1]]
449     else:
450         neighbors = np.array([grid[layer, j-1, i-1  ], grid[↩
                layer, j, i-1  ], grid[layer, j-L+1, i-1  ],
451                               grid[layer, j-1, i    ], grid[↩
                                  layer, j, i    ], grid[layer,↩
                                   j-L+1, i    ],
452                               grid[layer, j-1, i-L+1], grid[↩
                                  layer, j, i-L+1], grid[layer,↩
                                   j-L+1, i-L+1]])
453     if numLayers == 2:
454         layerNeighbors = np.asarray([grid[layer-1, j, i]])
455     else:
456         layerNeighbors = np.asarray([grid[layer-numLayers+1, j,↩
                i], grid[layer-1, j, i]])
457     allNeighbors = np.empty(len(neighbors)+len(layerNeighbors))↩
            .astype(np.float32)
458     allNeighbors[:len(neighbors)] = neighbors
459     allNeighbors[len(neighbors):] = layerNeighbors
460     return np.mean(allNeighbors)
461
462 #%%% Update Rules
463 @nb.njit()
464 def updateContGridBase(activationFunction, nbc, previous, grid,↩
        p1,p2,p3):
465     for j,i in gridCoordinates:
466         a_in = nbc(latticeSize, previous, j, i)
467         a_out = activationFunction(a_in, p1,p2,p3)
468         grid[j,i] = a_out
469     return grid
470
471 @nb.njit()
472 def updateContGridExternal(activationFunction, nbc, previous, ↩
        grid, p1,p2,p3, externalIndicesList):
473     for j,i in gridCoordinates:
474         a_in = nbc(latticeSize, previous, j, i)
475         a_out = activationFunction(a_in, p1,p2,p3)
476         grid[j,i] = a_out
477     grid = inject(grid, externalIndicesList)
478     return grid
```

```
479
480
481 @nb.njit()
482 def updateContGridLayered(activationFunction, nbc, previous, ↩
    grid, p1,p2,p3, numberLayers):
483     for layer in range(numberLayers):
484         for j,i in gridCoordinates:
485             a_in = nbc(latticeSize, previous, j, i, np.int32(↩
                numberLayers), np.int32(layer))
486             a_out = activationFunction(a_in, p1,p2,p3)
487             grid[layer,j,i] = a_out
488     return grid
489
490 @nb.njit()
491 def updateDiscGridBase(previousState, gridState, gridActivity, ↩
    gridProbability, previousRefractoryTime, gridRefractoryTime↩
    ):
492     for j,i in gridCoordinates:
493         gridState[j,i], gridRefractoryTime[j,i] = applyRQF(↩
            previousState[j,i], gridActivity[j,i], ↩
            gridProbability[j,i], previousRefractoryTime[j,i])
494     return gridState, gridRefractoryTime
495
496 @nb.njit()
497 def updateDiscGridLayered(previousState, gridState, ↩
    gridActivity, gridProbability, previousRefractoryTime, ↩
    gridRefractoryTime, numberLayers):
498     for layer in range(numberLayers):
499         for j,i in gridCoordinates:
500             gridState[layer,j,i], gridRefractoryTime[layer,j,i]↩
                = applyRQF(previousState[layer,j,i], ↩
                gridActivity[layer,j,i], gridProbability[j,i], ↩
                previousRefractoryTime[layer,j,i])
501     return gridState, gridRefractoryTime
502
503 @nb.njit()
504 def inject(grid, externalIndicesList):
505     for j,i in externalIndicesList:
506         grid[j,i] = 1
507     return grid
508
509 @nb.njit()
510 def applyRQF(state, activity, probability, refractoryTime):
511     if state == Q and probability <= activity:
512         return F, 0
513     elif state == F:
514         return R, 1
515     elif state == R and refractoryTime == 1:
516         return R, 2
```

```
517     elif state == R and refractoryTime == 2:
518         return Q, 0
519     else:
520         return Q, 0
521 
522 def applyCobweb(p1,p2,p3, initialState):
523     x0, y0 = initialState, 0
524     cobwebX, cobwebY = np.zeros(cobwebTimesteps*2+1), np.zeros(↩
        cobwebTimesteps*2+1)
525     cobwebX[0], cobwebY[0] = x0, y0
526     for t in range(cobwebTimesteps):
527         y0 = activationFunction(x0, p1,p2,p3)
528         cobwebX[2*t+1] = x0
529         cobwebY[2*t+1] = y0
530 
531         x0 = y0
532         cobwebX[2*t+2] = x0
533         cobwebY[2*t+2] = y0
534 
535     ### Getting steady-state and period
536     diffs_ = np.where(np.diff(cobwebX[::2])==0)[0]
537     if diffs_.size == 0:
538         steadyState, period = cobwebX[cobwebTimesteps*2], ↩
            cobwebTimesteps
539     else:
540         period = diffs_[0]
541         steadyState = cobwebX[period*2]
542     return cobwebX, cobwebY, steadyState, period
543 
544 #%% Spatiotemporal Analysis
545 
546 #%%% Continuous
547 def generateContSptm_raw(pathPrefix, parametersList, ↩
    automatonType, *, externalIndicesList=np.zeros([1]), ↩
    numberLayers=1):
548     if 'layered' in automatonType:
549         nbcFunctionsList = nbcLayeredFunctionsList
550         initialGrid = np.asarray([np.load(f'./lattice-{↩
                latticeSize}/initial-states/{layer}.npy') for layer↩
                in range(numberLayers)])
551     else:
552         nbcFunctionsList = nbcBaseFunctionsList
553         initialGrid = np.load(f'./lattice-{latticeSize}/initial↩
                -states/0.npy')
554     for nbc_id, nbc in tqdm(enumerate(nbcFunctionsList), desc=f↩
            'Generating {functionType} {automatonType} cont-sptm ↩
            raw'):
555         for p1, p2, p3 in parametersList:
556             grid = initialGrid.copy()
```

```
557             if 'external' in automatonType:     grid = inject(grid, externalIndicesList)
558             np.save(f'{pathPrefix}/spatiotemporal/{nbcList[nbc_id]}/{automatonType}/cont-sptm/raw/{p1}-{p2}-{p3}/0.npy', grid)
559             for t in range(DURATION):
560                 previous = grid.copy()
561                 if 'layered' in automatonType:      grid = updateContGridLayered(activationFunction, nbc, previous, grid, p1,p2,p3, numberLayers)
562                 elif 'external' in automatonType:   grid = updateContGridExternal(activationFunction, nbc, previous, grid, p1,p2,p3, externalIndicesList)
563                 else:                               grid = updateContGridBase(activationFunction, nbc, previous, grid, p1,p2,p3)
564                 np.save(f'{pathPrefix}/spatiotemporal/{nbcList[nbc_id]}/{automatonType}/cont-sptm/raw/{p1}-{p2}-{p3}/{t+1}.npy', grid)
565
566 def plotContSptm_png(pathPrefix, parametersList, automatonType, *, fractionExternal=0.0, numberLayers=1):
567     for nbc, [p1,p2,p3] in tqdm(it.product(nbcList, parametersList), desc=f'Plotting {functionType} {automatonType} cont-sptm png'):
568         filenames = sorted(glob.glob(f'{pathPrefix}/spatiotemporal/{nbc}/{automatonType}/cont-sptm/raw/{p1}-{p2}-{p3}/*.npy'), key=os.path.getmtime)
569         npy_dict  = {timeList[index]:np.load(filenames[index]) for index in timeList}
570         for t in timeList:
571             grid  = npy_dict[t]
572             if 'layered' in automatonType:      grid = np.hstack(tuple([grid[layer] for layer in range(numberLayers)]))
573             img   = Image.fromarray(np.uint8(sptcmap(grid)*255))
574             img.save(f'{pathPrefix}/spatiotemporal/{nbc}/{automatonType}/cont-sptm/png/{p1}-{p2}-{p3}/{t}.png')
575
576 def plotContSptm_gif(pathPrefix, parametersList, automatonType, *, fractionExternal=0.0, numberLayers=1):
577     for nbc, [p1,p2,p3] in tqdm(it.product(nbcList, parametersList), desc=f'Plotting {functionType} {automatonType} cont-sptm gif'):
```

```
578         path        = f'{pathPrefix}/spatiotemporal/{nbc}/{automatonType}/cont-sptm/png/{p1}-{p2}-{p3}'
579         fp_in       = f'{path}/*.png'
580         fp_out      = f'{pathPrefix}/spatiotemporal/{nbc}/{automatonType}/cont-sptm/gif/{p1}-{p2}-{p3}.gif'
581         img, *imgs = [Image.open(f) for f in sorted(glob.glob(fp_in), key=os.path.getmtime)]
582         img.save(fp=fp_out, format='GIF', append_images=imgs, save_all=True, duration=100, loop=0)
583
584 def generateContMean_raw(pathPrefix, parametersList, automatonType, *, fractionExternal=0.0, numberLayers=1):
585     for nbc, [p1,p2,p3] in tqdm(it.product(nbcList,parametersList), desc=f'Generating {functionType} {automatonType} cont-mean raw'):
586         filenames  = sorted(glob.glob(f'{pathPrefix}/spatiotemporal/{nbc}/{automatonType}/cont-sptm/raw/{p1}-{p2}-{p3}/*.npy'), key=os.path.getmtime)
587         dynamics   = np.asarray([np.mean(np.load(filenames[timestep])) for timestep in timeList])
588         if 'layered' in automatonType:
589             np.save(f'{pathPrefix}/spatiotemporal/{nbc}/{automatonType}/cont-mean/raw/{p1}-{p2}-{p3}-0.npy', dynamics)
590             for layer in range(numberLayers):
591                 dynamics = np.asarray([np.mean(np.load(filenames[timestep])[layer]) for timestep in timeList])
592                 np.save(f'{pathPrefix}/spatiotemporal/{nbc}/{automatonType}/cont-mean/raw/{p1}-{p2}-{p3}-{layer+1}.npy', dynamics)
593         else:   np.save(f'{pathPrefix}/spatiotemporal/{nbc}/{automatonType}/cont-mean/raw/{p1}-{p2}-{p3}.npy', dynamics)
594
595 def plotContMean_png(pathPrefix, parametersList, automatonType, *, fractionExternal=0.0, numberLayers=1):
596     plt.ioff()
597     fig, ax = plt.subplots(subplot_kw=dict(xlim=(-0.5, DURATION+0.5), ylim=(-0.005, 1.005), xlabel='Timestep', ylabel=r'Neuronal patch average $\langle a \rangle$'))
598     if 'layered' in automatonType:
599         zorder = numberLayers+1
600         for layer in range(zorder):
601             if layer == 0: legend_label='All Layers'
602             else: legend_label = f'Layer {layer}'
603             ax.plot(range(DURATION+1), np.empty(DURATION+1), color=colorList[layer], linewidth=zorder*2, marker='o', markersize=zorder*2, zorder=layer,
```

```
                label=legend_label)
604             zorder -= 1
605         plt.legend()
606     else:
607         line,    = ax.plot(range(DURATION+1), np.empty(DURATION+1), color='k', linewidth=2, marker='o', markersize=5)
608     for nbc, [p1,p2,p3] in tqdm(it.product(nbcList,parametersList), desc=f'Plotting {functionType} {automatonType} cont-mean png'):
609         ax.set_title(f'${parameterSymbolList[functionChoice][0]} = {p1} ; \
610                         {parameterSymbolList[functionChoice][1]} = {p2} ; \
611                         {parameterSymbolList[functionChoice][2]} = {p3}$' )
612         if 'layered' in automatonType:
613             for line_id, line in enumerate(ax.lines):
614                 dynamics = np.load(f'{pathPrefix}/spatiotemporal/{nbc}/{automatonType}/cont-mean/raw/{p1}-{p2}-{p3}-{line_id}.npy')
615                 line.set_ydata(dynamics)
616         else:
617             dynamics = np.load(f'{pathPrefix}/spatiotemporal/{nbc}/{automatonType}/cont-mean/raw/{p1}-{p2}-{p3}.npy')
618             line.set_ydata(dynamics)
619         plt.savefig(f'{pathPrefix}/spatiotemporal/{nbc}/{automatonType}/cont-mean/png/{p1}-{p2}-{p3}.png', bbox_inches='tight')
620     plt.close('all')
621
622 #%% Discrete
623 def generateDiscSptm_raw(pathPrefix, parametersList, automatonType, *, numberLayers=1):
624     for nbc, [p1,p2,p3] in tqdm(it.product(nbcList,parametersList), desc=f'Generating {functionType} {automatonType} disc-sptm raw'):
625         if 'layered' in automatonType:
626             gridState          = np.ones((numberLayers,latticeSize,latticeSize), dtype=int)
627             gridRefractoryTime = np.zeros((numberLayers,latticeSize,latticeSize), dtype= int)
628         else:
629             gridState          = np.ones((latticeSize,latticeSize), dtype=int)
630             gridRefractoryTime = np.zeros((latticeSize,latticeSize), dtype= int)
```

```
631         np.save(f'{pathPrefix}/spatiotemporal/{nbc}/{automatonType}/disc-sptm/raw/{p1}-{p2}-{p3}/0.npy', gridState)
632         for t in range(DURATION):
633             gridActivity = np.load(f'{pathPrefix}/spatiotemporal/{nbc}/{automatonType}/cont-sptm/raw/{p1}-{p2}-{p3}/{t+1}.npy')
634             gridProbability = np.load(f'./lattice-{latticeSize}/neuron-firing-probability/{t}.npy')
635             previousState = gridState.copy()
636             previousRefractoryTime = gridRefractoryTime.copy()
637             if 'layered' in automatonType:      gridState, gridRefractoryTime = updateDiscGridLayered(previousState, gridState, gridActivity, gridProbability, previousRefractoryTime, gridRefractoryTime, numberLayers)
638             else:                               gridState, gridRefractoryTime = updateDiscGridBase(previousState, gridState, gridActivity, gridProbability, previousRefractoryTime, gridRefractoryTime)
639             np.save(f'{pathPrefix}/spatiotemporal/{nbc}/{automatonType}/disc-sptm/raw/{p1}-{p2}-{p3}/{t+1}.npy', gridState)
640
641 def plotDiscSptm_png(pathPrefix, parametersList, automatonType, *, fractionExternal=0.0, numberLayers=1):
642     for nbc, [p1,p2,p3] in tqdm(it.product(nbcList, parametersList), desc=f'Plotting {functionType} {automatonType} disc-sptm png'):
643         filenames = sorted(glob.glob(f'{pathPrefix}/spatiotemporal/{nbc}/{automatonType}/disc-sptm/raw/{p1}-{p2}-{p3}/*.npy'), key=os.path.getmtime)
644         npy_dict  = {timeList[index]:np.load(filenames[index]) for index in timeList}
645         for t in timeList:
646             grid  = npy_dict[t]
647             if 'layered' in automatonType:      grid = np.hstack(tuple([grid[layer] for layer in range(numberLayers)]))
648             img   = Image.fromarray(np.uint8(RQFcmap(grid)*255))
649             img.save(f'{pathPrefix}/spatiotemporal/{nbc}/{automatonType}/disc-sptm/png/{p1}-{p2}-{p3}/{t}.png')
650
651 def plotDiscSptm_gif(pathPrefix, parametersList, automatonType, *, fractionExternal=0.0, numberLayers=1):
```

```
652     for nbc, [p1,p2,p3] in tqdm(it.product(nbcList,↩
            parametersList), desc=f'Plotting {functionType} {↩
            automatonType} disc-sptm gif'):
653         path       = f'{pathPrefix}/spatiotemporal/{nbc}/{↩
                automatonType}/disc-sptm/png/{p1}-{p2}-{p3}'
654         fp_in      = f'{path}/*.png'
655         fp_out     = f'{pathPrefix}/spatiotemporal/{nbc}/{↩
                automatonType}/disc-sptm/gif/{p1}-{p2}-{p3}.gif'
656         img, *imgs = [Image.open(f) for f in sorted(glob.glob(↩
                fp_in), key=os.path.getmtime)]
657         img.save(fp=fp_out, format='GIF', append_images=imgs, ↩
                save_all=True, duration=100, loop=0)
658
659 def generateDiscMean_raw(pathPrefix, parametersList, ↩
        automatonType, *, fractionExternal=0.0, numberLayers=1):
660     for nbc, [p1,p2,p3] in tqdm(it.product(nbcList,↩
            parametersList), desc=f'Generating {functionType} {↩
            automatonType} disc-mean raw'):
661         filenames    = sorted(glob.glob(f'{pathPrefix}/↩
                spatiotemporal/{nbc}/{automatonType}/disc-sptm/raw↩
                /{p1}-{p2}-{p3}/*.npy'), key=os.path.getmtime)
662         if 'layered' in automatonType:
663             for layer in range(numberLayers):
664                 dynamics   = np.asarray([np.load(filenames[↩
                        timestep])[layer, neuronIDy, neuronIDx] for↩
                         timestep in timeList])
665                 np.save(f'{pathPrefix}/spatiotemporal/{nbc}/{↩
                        automatonType}/disc-mean/raw/{p1}-{p2}-{p3↩
                        }/{p1}-{p2}-{p3}-{layer+1}.npy', dynamics)
666         else:
667             dynamics = np.asarray([np.load(filenames[timestep])↩
                    [neuronIDy, neuronIDx] for timestep in timeList↩
                    ])
668             np.save(f'{pathPrefix}/spatiotemporal/{nbc}/{↩
                    automatonType}/disc-mean/raw/{p1}-{p2}-{p3}.npy↩
                    ', dynamics)
669
670 def plotDiscMean_png(pathPrefix, parametersList, automatonType,↩
        *, fractionExternal=0.0, numberLayers=1):
671     plt.ioff()
672     fig, ax = plt.subplots(subplot_kw=dict(xlim=(-0.5, DURATION↩
            +0.5), ylim=(-0.25, 1.05)
673                            , xlabel='Timestep', ylabel='↩
                                Neuronal Discrete State'
674                            , xticks=[], xticklabels=[], yticks↩
                                =[], yticklabels=[]))
675     if 'layered' in automatonType:
676         zorder = numberLayers
677         for layer in range(zorder):
```

```
678             legend_label = f'Layer {layer+1}'
679             ax.plot(range(DURATION+1), np.empty(DURATION+1), ↩
                    color=colorList[layer], linewidth=zorder*2, ↩
                    marker='o', markersize=zorder*2, zorder=layer, ↩
                    label=legend_label)
680             zorder -= 1
681         plt.legend()
682     else:
683         line,    = ax.plot(range(DURATION+1), np.empty(DURATION↩
                +1), color='k', linewidth=2, marker='o', markersize↩
                =5)
684     for nbc, [p1,p2,p3] in tqdm(it.product(nbcList,↩
            parametersList), desc=f'Plotting {functionType} {↩
            automatonType} disc-mean png'):
685         ax.set_title(f'${parameterSymbolList[functionChoice↩
                ][0]} = {p1} ; \
686                         {parameterSymbolList[functionChoice][1]} =↩
                            {p2} ; \
687                         {parameterSymbolList[functionChoice][2]} =↩
                            {p3}$' )
688         if 'layered' in automatonType:
689             for line_id, line in enumerate(ax.lines):
690                 dynamics = np.load(f'{pathPrefix}/↩
                        spatiotemporal/{nbc}/{automatonType}/disc-↩
                        mean/raw/{p1}-{p2}-{p3}/{p1}-{p2}-{p3}-{↩
                        line_id+1}.npy')
691                 dynamics2 = [discreteResponseList[int(state)] ↩
                        for state in dynamics]
692                 line.set_ydata(dynamics2)
693         else:
694             dynamics = np.load(f'{pathPrefix}/spatiotemporal/{↩
                    nbc}/{automatonType}/disc-mean/raw/{p1}-{p2}-{↩
                    p3}.npy')
695             dynamics2 = [discreteResponseList[int(state)] for ↩
                    state in dynamics]
696             line.set_ydata(dynamics2)
697         plt.savefig(f'{pathPrefix}/spatiotemporal/{nbc}/{↩
                automatonType}/disc-mean/png/{p1}-{p2}-{p3}.png', ↩
                bbox_inches='tight')
698     plt.close('all')
699
700 #%% Local Analysis
701
702 #%%% Cobweb Diagram
703 def generateCobwebPoints_raw(pathPrefix, parametersList):
704     if parametersLength != 6:
705         initialStatesList = np.round(np.linspace(0, 1, ↩
                parametersLength), round_)
```

```
706     for p1, p2, p3 in tqdm(parametersList, desc=f'Generating {functionType} local cobweb diagram raw'):
707         for initialState in initialStatesList:
708             xcob, ycob, _, _ = applyCobweb(p1, p2, p3, initialState)
709             cobwebPoints = np.round(np.asarray([xcob,ycob]), cobwebPrecision)
710             np.save(f'{pathPrefix}/local/cobweb-diagram/raw/{p1}-{p2}-{p3}/{initialState}.npy', cobwebPoints)
711
712 def plotCobwebDiagram_png(pathPrefix, parametersList):
713     if parametersLength != 6:
714         initialStatesList = np.round(np.linspace(0, 1, parametersLength), round_)
715     plt.ioff()
716     cobwebDiagonal   = np.linspace(0,1,300)
717     cobwebX, cobwebY = np.zeros(cobwebTimesteps*2+1), np.zeros(cobwebTimesteps*2+1)
718     fig, ax      = plt.subplots(subplot_kw=dict(aspect='equal',
            xlim = (-0.03, 1.03), ylim = (-0.03, 1.03),
719                                                xlabel='Previous State',
                                                   ylabel='Next State'))
720     diagonal,    = ax.plot(cobwebDiagonal, cobwebDiagonal, color='k', linestyle ='--',zorder = 0)
721     activation,  = ax.plot(cobwebDiagonal, cobwebDiagonal, color='k', linewidth = 4, zorder=1)
722     initial,     = ax.plot(0, 0, color='b', marker='o', markersize=14, zorder = 3)
723     final,       = ax.plot(0, 0, color='g', marker='*', markersize=16, zorder = 3)
724     cobweb,      = ax.plot(cobwebX, cobwebY, color = 'r', linewidth = 3, zorder = 2)
725     for [p1, p2, p3], initialState in tqdm(it.product(parametersList, initialStatesList), desc=f'Plotting {functionType} local cobweb diagram png'):
726         activFuncY = [activationFunction(x_,p1,p2,p3) for x_ in cobwebDiagonal]
727         activation.set_ydata(activFuncY)
728         ax.set_title(f'${parameterSymbolList[functionChoice][0]} = {p1} ; \
729                         {parameterSymbolList[functionChoice][1]} = {p2} ; \
730                         {parameterSymbolList[functionChoice][2]} = {p3}$' )
```

```
731         cobwebPoints = np.load(f'{pathPrefix}/local/cobweb-↩
                diagram/raw/{p1}-{p2}-{p3}/{initialState}.npy')
732         cobweb.set_data(cobwebPoints)
733         initial.set_data(cobwebPoints[:,0])
734         final.set_data(cobwebPoints[:,len(cobwebPoints)-3])
735         plt.savefig(f'{pathPrefix}/local/cobweb-diagram/png/{p1↩
                }-{p2}-{p3}/{initialState}.png', bbox_inches='tight↩
                ')
736     plt.close('all')
737
738 #%% Fixed Point
739 def generateFixedPoints_raw(pathPrefix, parametersList, ↩
        lastTimesteps_PS_BD):
740     if parametersLength != 6:
741         initialStatesList = np.round(np.linspace(0, 1, ↩
                parametersLength), round_)
742     fixedPointSpace = np.zeros((2,lastTimesteps_PS_BD*↩
            parametersLength))
743     for p1, p2, p3 in tqdm(parametersList, desc=f'Generating {↩
            functionType} local fixed points raw'):
744         for isIndex, initialState in enumerate(↩
                initialStatesList):
745             dynamics = np.load(f'{pathPrefix}/local/cobweb-↩
                    diagram/raw/{p1}-{p2}-{p3}/{initialState}.npy')
746             fixedPointSpace[0, 0+10*isIndex:10*(isIndex+1)] = ↩
                    initialState
747             fixedPointSpace[1, 0+10*isIndex:10*(isIndex+1)] = ↩
                    dynamics[0, cobwebTimesteps*2+1-↩
                    lastTimesteps_PS_BD:]
748         np.save(f'{pathPrefix}/local/fixed-point-bifurcation/↩
                raw/{p1}-{p2}-{p3}.npy', fixedPointSpace)
749
750 def plotFixedPoints_png(pathPrefix, parametersList):
751     plt.ioff()
752     cobwebDiagonal   = np.linspace(0,1,300)
753     fpSpace          = np.zeros((2,110))
754     fig,ax = plt.subplots(subplot_kw=dict(aspect='equal',xlabel↩
            ='Initial State', ylabel='Steady-State'))
755     activation,  = ax.plot(cobwebDiagonal, cobwebDiagonal, ↩
            color='k', linewidth = 3, zorder=1)
756     diagonal,    = ax.plot(cobwebDiagonal, cobwebDiagonal, ↩
            color='gray', linestyle ='--',zorder = 0)
757     fpSpacePts,  = ax.plot(fpSpace[0, :], fpSpace[1,:], color='↩
            k', ls="None", marker='o', ms=8)
758     for p1, p2, p3 in tqdm(parametersList, desc=f'Plotting {↩
            functionType} local fixed points png'):
759         activFuncY = [activationFunction(x_,p1,p2,p3) for x_ in↩
                cobwebDiagonal]
760         activation.set_ydata(activFuncY)
```

```
761         fpSpace = np.load(f'{pathPrefix}/local/fixed-point-↩
                bifurcation/raw/{p1}-{p2}-{p3}.npy')
762         fpSpacePts.set_data(fpSpace)
763         ax.set_title(f'${parameterSymbolList[functionChoice↩
                ][0]} = {p1} ; \
764                         {parameterSymbolList[functionChoice][1]} =↩
                             {p2} ; \
765                         {parameterSymbolList[functionChoice][2]} =↩
                             {p3}$' )
766         plt.savefig(f'{pathPrefix}/local/fixed-point-↩
                bifurcation/png/{p1}-{p2}-{p3}.png', bbox_inches='↩
                tight')
767     plt.close('all')
768 
769 #%% Steady-State Bifurcation
770 def generateSSPhaseSpace_raw(pathPrefix, parametersList, ↩
        lastTimesteps_PS_BD):
771     if parametersLength != 6:
772         initialStatesList = np.round(np.linspace(0, 1, ↩
                parametersLength), round_)
773     phaseSpace = np.zeros((parametersLength,parametersLength), ↩
            dtype=np.float32)
774     tempList = np.zeros(parametersList.shape)
775     for bdIndex, (_p, _q) in tqdm(enumerate(reversed(↩
            phaseSpaceIndex)), desc='Generating {functionType} ↩
            local steady-state phase space raw'):
776         _r = (set([0, 1, 2]) - set([_p, _q])).pop()
777         tempList[:,0] = parametersList[:,_p]
778         tempList[:,1] = parametersList[:,_q]
779         tempList[:,2] = parametersList[:,_r]
780         tempList = tempList[np.lexsort((tempList[:,2], tempList↩
                [:,1], tempList[:,0]))]
781         for i_ in range(parametersLength*parametersLength):
782             phaseSpaceParameters = tempList[0+parametersLength*↩
                    i_:parametersLength*(i_+1),:]
783             for row, value in enumerate(phaseSpaceParameters):
784                 if bdIndex == 0:
785                     p2, p3, p1 = value
786                 elif bdIndex == 1:
787                     p1, p3, p2 = value
788                 elif bdIndex == 2:
789                     p1, p2, p3 = value
790                 for column, initialState in enumerate(↩
                        initialStatesList):
791                     dynamics = np.load(f'{pathPrefix}/local/↩
                            cobweb-diagram/raw/{p1}-{p2}-{p3}/{↩
                            initialState}.npy')
792                     phaseSpace[row, column] = np.mean(dynamics↩
                            [0,cobwebTimesteps*2+1-↩
```

```
                                lastTimesteps_PS_BD:])
793             np.save(f'{pathPrefix}/local/phase-space-steady-↩
                    state/raw/{bifurcationCaseList[bdIndex]}/{value↩
                    [0]}-{value[1]}.npy', phaseSpace)
794
795 def plotSSPhaseSpace_png(pathPrefix, parametersList):
796     phaseSpace = np.zeros((parametersLength,parametersLength), ↩
        dtype=np.float32)
797     ticks = np.asarray(range(parametersLength))[::2*round_]
798     xticklabels = np.round(np.linspace(0, 1, parametersLength),↩
         round_)[::2*round_]
799     plt.ioff()
800     fig, ax  = plt.subplots(subplot_kw=dict(aspect='equal', ↩
        xticks= ticks, yticks=ticks, xticklabels=xticklabels))
801     im = ax.imshow(phaseSpace, cmap = 'magma', vmin = 0, vmax↩
        =1)
802     for bdIndex, (_p, _q) in tqdm(enumerate(reversed(↩
        phaseSpaceIndex)), desc='Plotting {functionType} local ↩
        steady-state phase space png'):
803         _r = (set([0, 1, 2]) - set([_p, _q])).pop()
804         _pList = np.unique(parametersList[:,_p])
805         _qList = np.unique(parametersList[:,_q])
806         _rList = np.unique(parametersList[:,_r])
807         yticklabels = _rList[::2*round_]
808         ax.set(yticklabels=yticklabels, xlabel='Initial State',↩
                ylabel=f'{parameterNameList[functionChoice][_r]}, ↩
               ${parameterSymbolList[functionChoice][_r]}$')
809         for _p_, _q_ in it.product(_pList, _qList):
810             phaseSpace = np.load(f'{pathPrefix}/local/phase-↩
                    space-steady-state/raw/{bifurcationCaseList[↩
                    bdIndex]}/{_p_}-{_q_}.npy')
811             im.set_data(phaseSpace)
812             ax.set(title=f'${parameterSymbolList[functionChoice↩
                    ][_p]} = {_p_}$ ; ${parameterSymbolList[↩
                    functionChoice][_q]} = {_q_}$')
813             plt.savefig(f'{pathPrefix}/local/phase-space-steady↩
                    -state/png/{bifurcationCaseList[bdIndex]}/{_p_↩
                    }-{_q_}.png', bbox_inches='tight')
814     plt.close('all')
815
816 #%% Global Analysis
817
818 #%%% Phase Space
819 def generateGlobalPhaseSpace_raw(pathPrefix, parametersList, ↩
    automatonType, lastTimesteps_PS_BD):
820     phaseSpace = np.zeros((parametersLength,parametersLength), ↩
        dtype=np.float32)
821     tempList = np.zeros(parametersList.shape)
```

```
822     for nbc in tqdm(nbcList, desc=f'Generating {functionType} {automatonType} global phase space raw'):
823         for psIndex, (_q, _r) in enumerate(phaseSpaceIndex):
824             _p = (set([0, 1, 2]) - set([_q, _r])).pop()
825             tempList[:,0] = parametersList[:,_p]
826             tempList[:,1] = parametersList[:,_q]
827             tempList[:,2] = parametersList[:,_r]
828             tempList = tempList[np.lexsort((tempList[:,2], tempList[:,1], tempList[:,0]))]
829             for i_ in range(parametersLength):
830                 phaseSpaceParameters = tempList[0+parametersLength*parametersLength*i_:parametersLength*parametersLength*(i_+1), :]
831                 for coordIndex, (i,j) in enumerate(phaseSpaceCoordinates):
832                     if psIndex == 0:
833                         p3, p1, p2 = phaseSpaceParameters[coordIndex]
834                     elif psIndex == 1:
835                         p2, p1, p3 = phaseSpaceParameters[coordIndex]
836                     elif psIndex == 2:
837                         p1, p2, p3 = phaseSpaceParameters[coordIndex]
838                     dynamics = np.load(f'{pathPrefix}/spatiotemporal/{nbc}/{automatonType}/cont-mean/raw/{p1}-{p2}-{p3}.npy')
839                     phaseSpace[j, i] = np.mean(dynamics[DURATION+1-lastTimesteps_PS_BD:])
840                 np.save(f'{pathPrefix}/global/{nbc}/{automatonType}/phase-space/{phaseSpaceCaseList[psIndex]}/raw/{phaseSpaceParameters[0, 0]}.npy', phaseSpace)
841
842 def plotGlobalPhaseSpace_png(pathPrefix, parametersList, automatonType):
843     phaseSpace = np.zeros((parametersLength,parametersLength), dtype=np.float32)
844     ticks = np.asarray(range(parametersLength))[::2*round_]
845     plt.ioff()
846     fig, ax = plt.subplots(subplot_kw=dict(aspect='equal'))
847     im = ax.imshow(phaseSpace, cmap='magma', vmin=0, vmax=1) #,origin='lower')
848     for nbc in tqdm(nbcList, desc=f'Plotting {functionType} {automatonType} global phase space png'):
849         for psIndex, (_q, _r) in enumerate(phaseSpaceIndex):
850             _p = (set([0, 1, 2]) - set([_q, _r])).pop()
```

```
851             _pList = np.unique(parametersList[:,_p])
852             _qList = np.unique(parametersList[:,_q])[::2*round_↩
                    ]
853             _rList = np.unique(parametersList[:,_r])[::2*round_↩
                    ]
854             ax.set_xlabel(f'{parameterNameList[functionChoice][↩
                    _q]}, ${parameterSymbolList[functionChoice][_q↩
                    ]}$')
855             ax.set_ylabel(f'{parameterNameList[functionChoice][↩
                    _r]}, ${parameterSymbolList[functionChoice][_r↩
                    ]}$')
856             for _p_ in _pList:
857                 ax.set(xticks=ticks, xticklabels=_qList, yticks↩
                        =ticks, yticklabels=_rList)
858                 ax.set_title(f'${parameterSymbolList[↩
                        functionChoice][_p]} = {_p_}$')
859                 phaseSpace = np.load(f'{pathPrefix}/global/{nbc↩
                        }/{automatonType}/phase-space/{↩
                        phaseSpaceCaseList[psIndex]}/raw/{_p_}.npy'↩
                        )
860                 im.set_data(phaseSpace)
861                 plt.savefig(f'{pathPrefix}/global/{nbc}/{↩
                        automatonType}/phase-space/{↩
                        phaseSpaceCaseList[psIndex]}/png/{_p_}.png'↩
                        , bbox_inches='tight')
862     plt.close("all")
863
864 #%%% Bifurcation Diagram
865 def generateGlobalBifurcation_raw(pathPrefix, parametersList, ↩
    automatonType, lastTimesteps_PS_BD):
866     bifurcationSpace = np.zeros((lastTimesteps_PS_BD*↩
            parametersLength, 2))
867     tempList = np.zeros(parametersList.shape)
868     for nbc in tqdm(nbcList, desc=f'Generating {functionType} {↩
            automatonType} global bifurcation diagram raw'):
869         for bdIndex, (_p, _q) in enumerate(reversed(↩
                phaseSpaceIndex)):
870             _r = (set([0, 1, 2]) - set([_p, _q])).pop()
871             tempList[:,0] = parametersList[:,_p]
872             tempList[:,1] = parametersList[:,_q]
873             tempList[:,2] = parametersList[:,_r]
874             tempList = tempList[np.lexsort((tempList[:,2], ↩
                    tempList[:,1], tempList[:,0]))]
875             for i_ in range(parametersLength*parametersLength):
876                 bifurcationSpaceParameters = tempList[0+↩
                        parametersLength*i_:parametersLength*(i_+1)↩
                        , :]
877                 for bsIndex in range(parametersLength):
878                     if bdIndex == 0:
```

```
879                         p2, p3, p1 = bifurcationSpaceParameters↩
                                [bsIndex]
880                     elif bdIndex == 1:
881                         p1, p3, p2 = bifurcationSpaceParameters↩
                                [bsIndex]
882                     elif bdIndex == 2:
883                         p1, p2, p3 = bifurcationSpaceParameters↩
                                [bsIndex]
884                     dynamics = np.load(f'{pathPrefix}/↩
                            spatiotemporal/{nbc}/{automatonType}/↩
                            cont-mean/raw/{p1}-{p2}-{p3}.npy')
885                     bifurcationSpace[0+10*bsIndex:10*(bsIndex↩
                            +1), 0] = bifurcationSpaceParameters[↩
                            bsIndex, 2]
886                     bifurcationSpace[0+10*bsIndex:10*(bsIndex↩
                            +1), 1] = dynamics[DURATION+1-↩
                            lastTimesteps_PS_BD:]
887                 np.save(f'{pathPrefix}/global/{nbc}/{↩
                        automatonType}/bifurcation/{↩
                        bifurcationCaseList[bdIndex]}/raw/{↩
                        bifurcationSpaceParameters[bsIndex, 0]}-{↩
                        bifurcationSpaceParameters[bsIndex, 1]}.npy↩
                        ', bifurcationSpace)
888
889 def plotGlobalBifurcation_png(pathPrefix, parametersList, ↩
        automatonType):
890     plt.ioff()
891     for nbc in tqdm(nbcList, desc=f'Plotting {functionType} {↩
            automatonType} global bifurcation diagram png'):
892         for bdIndex, (_p, _q) in enumerate(reversed(↩
                phaseSpaceIndex)):
893             _r = (set([0, 1, 2]) - set([_p, _q])).pop()
894             _pList = np.unique(parametersList[:,_p])
895             _qList = np.unique(parametersList[:,_q])
896             _rList = np.unique(parametersList[:,_r])
897             for _p_, _q_ in it.product(_pList, _qList):
898                 fig, ax = plt.subplots(subplot_kw=dict(
899                                         xticks=_rList[::2*round_↩
                                            ], yticks=↩
                                            referenceList[::2*↩
                                            round_],
900                                         xlim=(-0.05, max(_rList)↩
                                            +0.05), ylim=(-0.05,↩
                                             1.05),
901                                         xlabel=f'{↩
                                            parameterNameList[↩
                                            functionChoice][_r↩
                                            ]}, ${↩
                                            parameterSymbolList[↩
```

```
                                                functionChoice][_r]}↩
                                                $', ylabel=f'Steady-↩
                                                States at $ t \\↩
                                                rightarrow \\infty $↩
                                                ',
902                                             title=f'${↩
                                                parameterSymbolList[↩
                                                functionChoice][_p]}↩
                                                 = {_p_}$ ; ${↩
                                                parameterSymbolList[↩
                                                functionChoice][_q]}↩
                                                 = {_q_}$'))
903                 bifurcationSpace = np.load(f'{pathPrefix}/↩
                        global/{nbc}/{automatonType}/bifurcation/{↩
                        bifurcationCaseList[bdIndex]}/raw/{_p_}-{↩
                        _q_}.npy')
904                 ax.plot(bifurcationSpace[:,0], bifurcationSpace↩
                        [:,1], color='k', ls="None", marker='o', ms↩
                        =8)
905                 ax.set_box_aspect(1)
906                 plt.savefig(f'{pathPrefix}/global/{nbc}/{↩
                        automatonType}/bifurcation/{↩
                        bifurcationCaseList[bdIndex]}/png/{_p_}-{↩
                        _q_}.png', bbox_inches='tight')
907                 plt.close("all")
908
909
910
911 #%% Default
912 if __name__ == "__main__":
913     with open('tqdm-log.txt', 'w') as LOGFILE:
914         main(
915             make_directory=True,
916             make_base=True,
917             make_external=True,
918             make_layered=True,
919             make_global=True,
920             make_local=True
921              )
```

Bibliography

[1] G. Cimino, "Reticular theory versus neuron theory in the work of Camillo Golgi.," Physis; rivista internazionale di storia della scienza **36,** 431–472 (1999).

[2] I. Mishqat, "The Formation of Reticular Theory," in *Embryo Project Encyclopedia* (Arizona State University, School of Life Sciences, Center for Biology and Society, The Embryo Project at Arizona State University, 1711 South Rural Road, Tempe Arizona 85287, United States, 2017), last accessed: 6-May-2022.

[3] I. Mishqat, "The Neuron Doctrine (1860-1895)," in *Embryo Project Encyclopedia* (Arizona State University, School of Life Sciences, Center for Biology and Society, The Embryo Project at Arizona State University, 1711 South Rural Road, Tempe Arizona 85287, United States, 2017), last accessed: 2022-05-06.

[4] S. Finger, *Origins of neuroscience: a history of explorations into brain function* (Oxford University Press, USA, 2001).

[5] G. M. Shepherd, *Foundations of the neuron doctrine* (Oxford University Press, 2015).

[6] NobelPrize.org, "The Nobel Prize in Physiology or Medicine 1906,", https://www.nobelprize.org/prizes/medicine/1906/summary/, last accessed: 2022-05-06.

[7] A. L. Hodgkin and A. F. Huxley, "A quantitative description of membrane current and its application to conduction and excitation in nerve," J. Physiol. **117,** 500–544 (1952).

[8] W. Gerstner, W. M. Kistler, R. Naud, and L. Paninski, *Neuronal Dynamics: From single neurons to networks and models of cognition and beyond* (Cambridge University Press, Cambridge, England, 2014), https://neuronaldynamics.epfl.ch/online/index.html.

[9] NobelPrize.org, "The Nobel Prize in Physiology or Medicine 1963,", https://www.nobelprize.org/prizes/medicine/1963/summary/, last accessed: 2022-05-06.

[10] W. H. Calvin, "Synaptic potential summation and repetitive firing mechanisms: Input-output theory for the recruitment of neurons into epileptic bursting firing patterns," Brain Research **39,** 71–94 (1972).

[11] D. Purves *et al.*, "Summation of Synaptic Potentials," Neuroscience. 2nd Edition. Sunderland, MA: Sinnauer Associates Inc (2001).

[12] A. Giannari and A. Astolfi, "Model design for networks of heterogeneous Hodgkin–Huxley neurons," Neurocomputing **496,** 147–157 (2022).

[13] L. Jennes, "Chapter 1 - Cytology of the Central Nervous System," in *Conn's Translational Neuroscience*, P. M. Conn, ed., (Academic Press, San Diego, 2017), pp. 1–10.

[14] "Brain Basics: The Life and Death of a Neuron,", https://www.ninds.nih.gov/health-information/patient-caregiver-education/brain-basics-life-and-death-neuron, last accessed: 2022-05-06.

[15] C. S. Breathnach, "Charles Scott Sherrington's Integrative action: A

centenary notice," Journal of the Royal Society of Medicine **97,** 34–36 (2004).

[16] J. H. Byrne, "Introduction to neurons and neuronal networks," Textbook for the Neurosciences p. 12 (2013).

[17] D. Purves *et al.*, "The organization of the nervous system," Neuroscience. 2nd Edition. Sunderland, MA: Sinnauer Associates Inc (2001).

[18] F. A. Azevedo, L. R. Carvalho, L. T. Grinberg, J. M. Farfel, R. E. Ferretti, R. E. Leite, W. J. Filho, R. Lent, and S. Herculano-Houzel, "Equal numbers of neuronal and nonneuronal cells make the human brain an isometrically scaled-up primate brain," Journal of Comparative Neurology **513,** 532–541 (2009).

[19] R. D. Fields, A. Araque, H. Johansen-Berg, S.-S. Lim, G. Lynch, K.-A. Nave, M. Nedergaard, R. Perez, T. Sejnowski, and H. Wake, "Glial biology in learning and cognition," The neuroscientist **20,** 426–431 (2014).

[20] K. R. Jessen and R. Mirsky, "Glial cells in the enteric nervous system contain glial fibrillary acidic protein," Nature **286,** 736–737 (1980).

[21] J. C. Waymire, "Neuroscience Online Chapter 8: Organization of Cell Types," in *Neuroscience Online Chapter 8: Organization of Cell Types* (The University of Texas Health Science Center at Houston, 1997).

[22] E. R. Kandel *et al.*, *Principles of neural science* (McGraw-hill New York, 2000), Vol. 4.

[23] "Neurons and Support Cells,", https://histology.siu.edu/ssb/neuron.htm, last accessed: 2022-05-06.

[24] N. Brunel and M. C. Van Rossum, "Quantitative investigations of electrical nerve excitation treated as polarization," Biological Cybernetics **97,** 341–349 (2007).

[25] D. Tal and E. Schwartz, "Computing with the leaky integrate-and-fire neuron: logarithmic computation and multiplication," Neural Comput. **9,** 305–318 (1997).

[26] J. C. S. Pang, Master's thesis, National Institute of Physics, College of Science, University of the Philippines, Diliman, Quezon City, 2014.

[27] J. C. S. Pang and J. Y. Bantang, "Hodgkin–Huxley neurons with defective and blocked ion channels," International Journal of Modern Physics C **26,** 1550112 (2015).

[28] S. A. Prescott, Y. De Koninck, and T. J. Sejnowski, "Biophysical basis for three distinct dynamical mechanisms of action potential initiation," PLoS computational biology **4,** e1000198 (2008).

[29] R. X. A. Ramos, J. C. Dominguez, and J. Y. Bantang, "Young and Aged Neuronal Tissue Dynamics With a Simplified Neuronal Patch Cellular Automata Model," Frontiers in Neuroinformatics **15,** 76 (2022).

[30] R. Ananthanarayanan, S. K. Esser, H. D. Simon, and D. S. Modha, "The Cat is out of the Bag: Cortical Simulations with 10^9 Neurons, 10^{13} Synapses," In *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, SC '09 pp. 1–12 (Association for Computing Machinery, New York, NY, USA, 2009).

[31] D. S. Bassett and M. S. Gazzaniga, "Understanding complexity in the human brain," Trends in cognitive sciences **15,** 200–209 (2011).

[32] J. von Neumann, *Theory of self-reproducing automata* (University of Illinois Press, USA, 1966).

[33] C. G. Langton, "Self-reproduction in cellular automata," Physica D: Nonlinear Phenomena **10,** 135–144 (1984).

[34] J. R. Pasta and S. Ulam, "Heuristic numerical work in some problems of hydrodynamics," Mathematical tables and other aids to computation **13,** 1–12 (1959).

[35] J. Conway *et al.*, "The game of life," Scientific American **223,** 4 (1970).

[36] S. Wolfram, *A New Kind of Science* (Wolfram Media, Inc., Champaign, Illinois, 2002), https://www.wolframscience.com/nks/.

[37] M. Arciaga, M. Pastor, R. Batac, J. Bantang, and C. Monterola, "Experimental observation and an empirical model of enhanced heap stability resulting from the mixing of granular materials," Journal of Statistical Mechanics: Theory and Experiment **2009,** P07040 (2009).

[38] R. X. Ramos and J. Bantang, "Totalistic cellular automata model of a neuronal network on a spherical surface," In *Proceedings of the Samahang Pisika ng Pilipinas*, **37,** SPP–2019–PB–16 (2019).

[39] R. Ramos and J. Bantang, *16th International Conference on Molecular Systems Biology* (De La Salle University, Manila, 2019), p. 51, poster presentation, https://drive.google.com/file/d/1bs655p68uv0SpZmmOcDr8MFV-_bWLvo_/view.

[40] R. Ramos and J. Bantang, *Brain Connects 2019 and the 9th Neuroscience International Symposium* (Global City, Taguig, 2019), pp. BC2019–14, poster and oral presentation.

[41] K. Hawick and C. Scogings, "Cycles, transients, and complexity in the game of death spatial automaton," In *Proc. International Conference on Scientific Computing (CSC'11)*, pp. 241–247 (2011).

[42] G. A. Wainer, "An introduction to cellular automata models with cell-DEVS," In *2019 Winter Simulation Conference (WSC)*, pp. 1534–1548 (2019).

[43] R. Ramos and J. Bantang, "Proposed cellular automaton model for a neuronal patch with a thresholded linear activation function," In *Proceedings of the Samahang Pisika ng Pilipinas*, **36,** SPP–2018–PB–37 (2018).

[44] R. X. A. Ramos, Bachelor's thesis, National Institute of Physics, College of Science, University of the Philippines, Diliman, Quezon City, 2019.

[45] E. W. Weisstein, "Logistic map," https://mathworld. wolfram. com/ (2001).

[46] M. Ausloos, *The logistic map and the route to chaos: From the beginnings to modern applications* (Springer Science & Business Media, 2006).

[47] A. Avila and C. G. Moreira, "Statistical properties of unimodal maps: the quadratic family," Annals of mathematics pp. 831–881 (2005).

[48] R. Stoop and W.-H. Steeb, *Berechenbares Chaos in dynamischen Systemen* (Springer-Verlag, 2006).

[49] R. X. A. Ramos and J. Y. Bantang, "Verhulst and bifurcation analyses of a neuronal network on an outer-totalistic toroidal cellular automata," In *Proceedings of the Samahang Pisika ng Pilipinas*, **38,** SPP–2020–3C–05 (2020).

[50] R. X. A. Ramos and J. Y. Bantang, "Simplified cellular automata model of neuronal patch dynamics with generalized non-linear cell response," In *Proceedings of the Samahang Pisika ng Pilipinas*, **39,** SPP–2021–PB–03 (2021).

[51] C. Çelik Karaaslanlı, "Bifurcation Analysis and Its Applications," in *Numerical Simulation: From Theory to Industry*, M. Andriychuk, ed., (INTECH Open Access Publisher, Pidstryhach Institute for Applied

Problems of Mechanics and Mathematics, Ukraine, 2012), Chap. 1, pp. 3–34.

[52] P. Ashwin, S. Coombes, and R. Nicks, "Mathematical frameworks for oscillatory network dynamics in neuroscience," The Journal of Mathematical Neuroscience **6,** 1–92 (2016).

[53] P. J. Coskren, J. I. Luebke, D. Kabaso, S. L. Wearne, A. Yadav, T. Rumbell, P. R. Hof, and C. M. Weaver, "Functional consequences of age-related morphologic changes to pyramidal neurons of the rhesus monkey prefrontal cortex," Journal of computational neuroscience **38,** 263–283 (2015).

[54] A. Peters, "The Effects of Normal Aging on Nerve Fibers and Neuroglia in the Central Nervous System," in *Brain Aging* (CRC Press, 2007), pp. 97–126.

[55] A. D. Rivera, F. Pieropan, I. Chacon-De-La-Rocha, D. Lecca, M. P. Abbracchio, K. Azim, and A. M. Butt, "Functional genomic analyses highlight a shift in Gpr17-regulated cellular processes in oligodendrocyte progenitor cells and underlying myelin dysregulation in the aged mouse cerebrum," Aging cell **20,** e13335 (2021).

[56] E. Pannese, "Morphological changes in nerve cells during normal aging," Brain Structure and Function **216,** 85–89 (2011).

[57] R. Mostany, J. E. Anstey, K. L. Crump, B. Maco, G. Knott, and C. Portera-Cailliau, "Altered synaptic dynamics during normal brain aging," Journal of Neuroscience **33,** 4094–4104 (2013).