Neural Profiles of Two- and Three-State Cellular Automata on Two-Dimensional Lattice Domains

By

RICHARD PAUL YIM

THESIS

Submitted in partial satisfaction of the requirements for the degree of

MASTER OF SCIENCE

in

APPLIED MATHEMATICS

in the

OFFICE OF GRADUATE STUDIES

of the

UNIVERSITY OF CALIFORNIA

DAVIS

Approved:

Janko Gravner, Chair

Niels Grønbech-Jensen

Alexander Wein

Committee in Charge

2023

To my friends and family.

Contents

Abstract iv

Acknowledgments v

Chapter 1. Introduction 1

1.1. Cellular Automata 1

1.2. Problem Formulation 4

1.3. Description of Chapters 4

Chapter 2. Neural Profiles of Cellular Automata 6

2.1. Cellular Automata on Lattices 6

2.2. Graph Convolutional Networks 10

2.3. Modeling Cellular Automata Rules 17

Chapter 3. Spectral Analysis of Cellular Automata Neural Profiles 25

3.1. Damage Spreading 25

3.2. Spectral Analysis of Hidden Layer Weight Matrices 28

Chapter 4. Conclusion 37

4.1. Summary of Results 37

4.2. Remarks 38

Appendix A. Additional Figures 40

A.1. Average Damage Spreading Dynamics 40

A.2. Empirical Spectral Analysis 41

Bibliography 43

Neural Profiles of Two- and Three-State Cellular Automata on Two-Dimensional Lattice Domains

Abstract

Neural profiles of cellular automata provide a systematic, computational framework for numerically and statistically characterizing seemingly disparate cellular automata transition rules. This computational framework utilizes graph convolution networks (GCNs) to exactly and precisely learn the full lookup table associated with a given cellular automaton rule. We address three hypotheses all centered on addressing the precision and accuracy of intrinsically characterizing cellular automata transition rules independent of any extrinsic phenomenological behaviors conventionally studied in the literature. This thesis comprises of two main parts: the first part formally specifies the process of fitting graph convolutional networks and designing datasets of cellular automata for GCNs to learn; the second part then defines various methods for analyzing GCNs corresponding to a given cellular automaton rule.

Acknowledgments

This thesis would not have been a fraction of what it has become had it not been for Professor Janko Gravner. I am deeply indebted to him for encouraging me and being supportive throughout the writing and development of this thesis. For the many hours of time and incredible thoughtfulness in advice related to this thesis and otherwise, I will be forever grateful and never forget his attention, care and kindness throughout the last two years.

I must also deeply thank Professor Deanna Needell during my time as an undergraduate at UCLA. Her support in my undergraduate years in research and teaching gave me the motivation to stay curious as well as a purpose to do anything at all during a very difficult and challenging last two years of undergrad. It was an incredible privilege that Professor Needell let me enroll in her numerical linear algebra course, which is where it all started.

In the same vein, I must thank Professor Hanbaek Lyu as he was the first person to get me into dynamical systems research. Summer 2020 was a very exciting time for me thanks to Han, and I never imagined that I would have the privilege to continue working with him throughout the last two years, during my master's. I am grateful for all the the time and care he has provided as a mentor in research especially while being hundreds of miles away.

I'm grateful that I was able to work closely with Hardeep Bassi for L2PSync. Countless hours of shared frustration and relief will always serve as a reminder as to why I enjoy scientific research: collaboration and camaraderie. I wish him continued success throughout his PhD and beyond.

I am also indebted to the UC Davis Mathematics Department. It was a privilege working as TA and receiving funding in my second year. I'd like to thank Tina Denena, Vanessa Bravo and Diana Bond for administrative support. To Professors Niels Grønbech-Jensen and Alex Wein, I'd like to thank them for being on my thesis committee. UC Davis has so many great professors and lecturers—it was such a privilege to be allowed to complete my MS degree at UC Davis—I learned so much and, fortunately, still not enough!

To Sihua, of course, I'm incredibly thankful for all the moral support she's given me in completing my degree and thesis, as well as tolerating all my ramblings about this thesis.

Lastly, to my parents: thank you for supporting my education and giving me your unconditional love and care. I certainly would not have made it this far without you.

CHAPTER 1

Introduction

1.1. Cellular Automata

Cellular automata (CA) are models of computation that can exhibit great complexity and organization from simple sets of rules. They have been of great interest since their conception [**1**] as evidenced by many variations of their analysis and published work. Elementary cellular automata (ECAs) are one type of CA operating specifically on a single dimensional lattice. Each cell of an ECA has values either 0 or 1, and is evolved in each time step according to a rule based on the states of its nearest neighbors, or adjacent cells. For instance, for a fixed neighborhood at time t, (L, C, R)—left, center and right cells—, the new state at time step $t + 1$ is determined via a lookup table based on states of the neighborhood of the center node, C [**2**]. There are 2^3 possible configurations for a binary string of length three representing the neighborhood of a center cell. Further, for each neighborhood configuration, there are two possible outcomes that can be assigned by any given rule; therefore we can fully enumerate all 2^8 possible rules that fall under the classification of an "elementary cellular automaton." Figure 1.1 shows diagrams of various dynamics produced by ECA rules and Table 1.1 shows an example ECA rule as a *lookup table* mapping neighborhood state configurations to updates for the center cell, C, at time $t + 1$.

TABLE 1.1. Example ECA corresponding to rule 90, or 01011010_2 as an eight character binary string corresponding to 8 possible permutations of binary characters. Notice that the first row corresponds to binary representations of 7 through 0, left to right, and the second row corresponds to the characters of the binary representation of rule 90 from left to right. The first row corresponds to the input neighbor of the $(L, C, R)_t$ neighborhood of the center cell at time t with corresponding output of the center cell at time $t + 1$.

Decimal Character	7	6	5	4	3	2	1	0
$(L, C, R)_t$	111	110	101	100	011	010	001	000
$(C)_{t+1}$	0	1	0	1	1	0	1	0

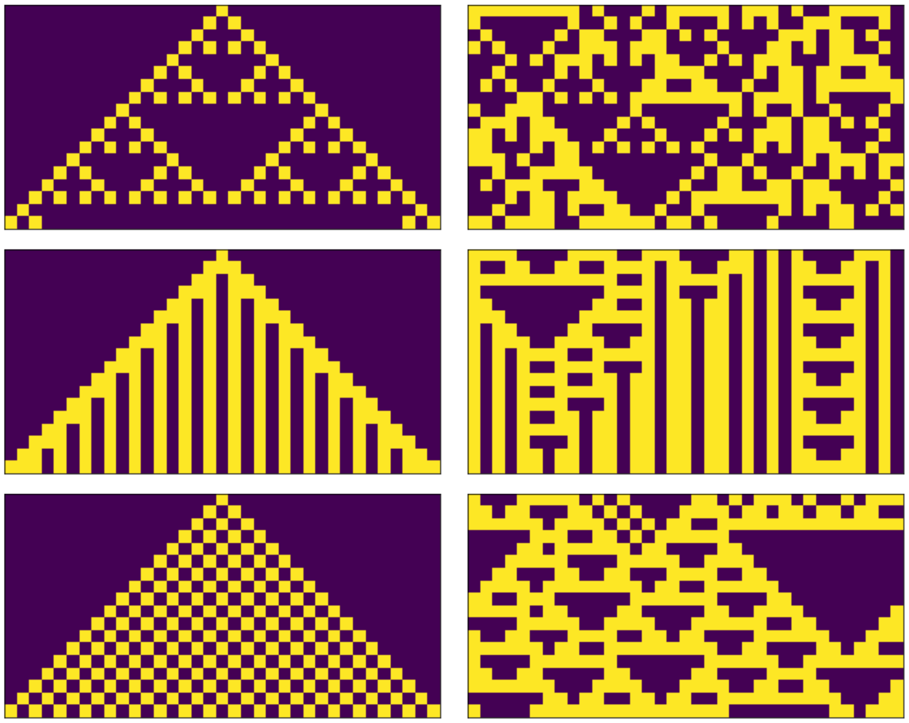

Figure 1.1. Each row in each diagram corresponds to 0 and 1 states in the length 35 one-dimensional array, or lattice, and these arrays are stacked in order from time 0 to time 17. Top-to-bottom are diagrams produced by ECA rules 90, 94 and 122. The first column of figures correspond to ECA dynamics produced on a length 35 array with the center cell, zero-index position 17, having a 1-state as initial configuration; the second column corresponds to random initial configurations. Most interesting is rule 90, which produces Sierpińsky triangles.

There are more sophisticated CA that operate on more than just a one-dimensional lattice. Even more famous than ECA is the two-dimensional cellular automaton, Conway's Game of Life (GOL). GOL operates on a two-dimensional lattice where each vertex, or cell, on the lattice has two possible states, 0 or 1. Like ECAs the update rule evolving a given state at time t works by observing the status of a cell's neighborhood, where the neighborhood of a given cell is a 3-by-3 grid with the target cell to be updated, located in the center.

In particular, GOL has the following update rule based on its neighbors' states and its own state:

1. Any 1-valued cell with two or three 1-valued neighbors remains 1 valued.
2. Any 0-valued cell with three 1-valued neighbors remains becomes 1 valued.

3. Otherwise, 1-valued cells become 0 valued; and 0-valued cells remain 0 valued.

Based on only these three rules, much complexity is exhibited visually from various initial configurations of CA—Figure 1.2 shows an instance of complex behavior through *gliders*.

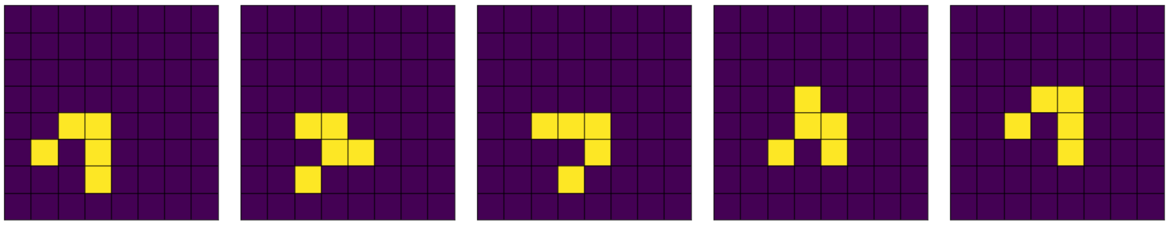

Figure 1.2. Gliders are one of many types of "vehicles" that emerge from Conway's Game of Life. The glider alternates between two shapes, up to a rotation and reflection, and becomes displaced, returning to the starting glider configuration and orientation every 5 steps.

We have identified lattice-based CA, but there are extensions of CA to more general noneuclidean domains—in particular, graphs. For example, the firefly cellular automata (FCA) is a CA that evolves κ-states periodically on $\mathbb{Z}/\kappa\mathbb{Z}$ over arbitrary graphs, including lattices. An inhibitory state is defined for the FCA as $b(\kappa) = \lfloor \frac{\kappa - 1}{2} \rfloor$, where the following update rule is assigned for each vertex, v, on an arbitrary graph domain:

$$X_{t+1}(v) = \begin{cases} X_t(v), \text{ if } X_t(v) > b(\kappa) \text{ and } \exists u \in N(v) \text{ s.t. } u = b(\kappa) \\ (X_t(v) + 1) \mod \kappa, \text{ otherwise} \end{cases},$$

where $N(v)$ denotes the first-order neighborhood of v. (Note that this inhibitory state is denoted as $b(\kappa)$ as in a blinking color with relation to the *blinking* lights used for communication between real fireflies.)

Many different characterizations of CA models have been developed and rigorously studied: Lyapunov profiles, symbolic dynamics and more [**3**] [**4**]. However, there has been limited work towards developing a single framework to study CA rules.

1.2. Problem Formulation

Most CA literature is often focused on analyzing a fixed CA model over a fixed domain and thoroughly studying associated emergent phenomena such as synchronization [**5**] [**6**] [**7**]. Furthermore, classifying different CA themselves are a very difficult task in and of themselves—Wolfram's classification is an imprecise example of such an attempt [**8**]. Further, little work exists in the way of intrinsically characterizing CA rules in a single topological space—a theory of cellular automata. While it is true that a general CA theory may be "out of the question," this thesis presents a formal computational framework to characterize various cellular automata through a single model with particular focus on CA over two-dimensional lattice domains [**9**].

This computational framework utilizes graph convolutional networks (GCNs) in combination with empirical random matrix theory to define what we call the *neural profile* of a cellular automaton. In particular, the neural profile of a cellular automaton refers to a distributional characterization of the update/transition rule associated with the cellular automaton. With this focus three main hypotheses are considered:

(**H1**) Neural profiles of CA are dependent on the parameter complexity of GCNs.

(**H2**) Distinct CA rules do not have different neural profile characteristics.

(**H3**) There is no association between CA stability and neural profile characteristics.

Beyond these hypotheses, the primary contribution of this thesis is the introduction of a well-specified and well-designed method for numerically characterizing seemingly disparate types of CA rules. This method will enable an intrinsic characterization of CA rules without relying on any extrinsic, phenomenological display of emergent behavior governed by those rules.

1.3. Description of Chapters

With the hypotheses defined for this thesis, we present the method and its results in two main chapters:

Chapter 2 will introduce the CA considered in this thesis as well as the background of GCNs. It will then describe the method for modeling CA lookup tables with GCNs. The subsequent sections in the chapter will present the formal process for constructing and fitting neural profiles of any given cellular automaton. This chapter will also address hypothesis **H1**.

Chapter 3 will further develop our model of CA rules by describing the tools used to analyze these neural profiles of cellular automata rules. It will introduce a characterization of cellular automata rules as learned by GCNs. In particular, this chapter will directly address hypotheses **H2** and **H3**.

Chapter 4 will conclude this thesis with a summary of all results and conclusions of the aforementioned hypotheses. It will also discuss current limitations and possible extensions of this method of neural profiles as well as possible directions for future works.

CHAPTER 2

Neural Profiles of Cellular Automata

2.1. Cellular Automata on Lattices

This section serves as a background on the various CA modeled with neural profiles. Relevant literature for each CA model will be discussed in addition to example dynamics produced by each rule according to some initial configuration over some domain. Details of the variations of CA considered in this project are provided in following subsections. Since this project primarily considers CA models on two-dimensional lattices we note that we solely consider two examples of neighborhoods. Namely, we consider lattices where each node—as arranged on a lattice—is paired with its 8 neighbors, a *Moore* neighborhood, as well as a 4 neighbor variation, a *von Neumann* neighborhood. With these different neighborhoods we focus on fixing a neural network to learn lookup tables corresponding to the transition function or rule of a cellular automaton—this is what we will occasionally refer to as *learning the rule*. Also, we primarily consider *totalistic*, or neighborhood-averaging type cellular automata [**10**].

2.1.1. Bootstrap Percolation (BSP). Bootstrap percolation (BSP) is not typically considered to be a CA model, yet it is precisely defined as type of cellular automaton. Compared to the following other CA models to follow, bootstrap percolation does not oscillate between states. Further, BSP only has two states, 0 and 1, such that states on the lattice all transition towards a 1 state or remain in a 0 state—no 1-state node becomes a 0-state node (we will refer to vertices on the lattice as *nodes* for the remainder of the thesis except in a graph theoretic context). What is unique about bootstrap percolation is that there is a threshold parameter τ that determines whether a given node on the lattice will be able to transition to a 1-state—essentially, the threshold governs whether or not a given node will be bootstrapped by its local neighborhood to a 1-state assuming it is initially at a 0-state. BSP is of particular interest to probability theorists where various conditions on the mode of convergence for percolation on the lattice are proven [**11**] [**12**]. Figure 2.1

shows example dynamics of this percolation phenomenon occurring from a random initial density of 70% 1-state cells on a lattice with periodic boundary. Below is a formal functional expression of state transition for a given node v, or vertex, on a lattice, along with $\mathcal{N}(v)$ corresponding to the set of neighbors of v; the notation $X_t(v)$ corresponds to the state of node v at time t:

$$X_{t+1}(v) = \begin{cases} 1, \text{ if } \sum_{u \in \mathcal{N}(v)} X_t(u) > \tau \text{ and } X_t(v) = 0 \\ 0, \quad \text{otherwise} \end{cases} .$$

(The function notation $X_{t+1}(v)$ will occasionally be used throughout the rest of the text when referring to "transition rules" or "lookup tables.")

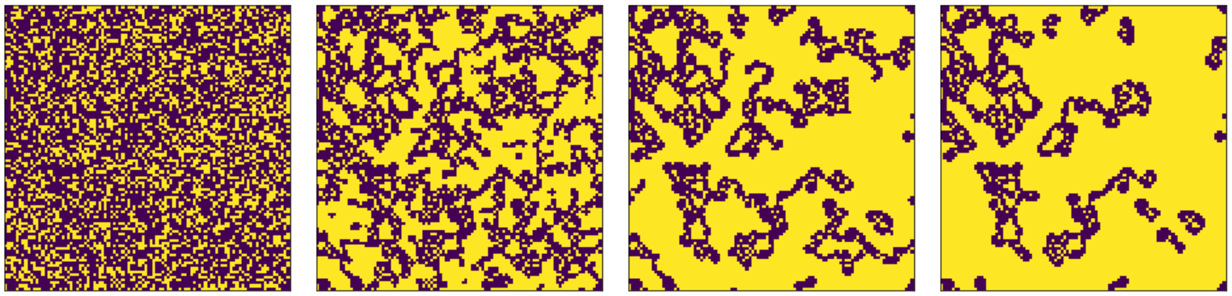

FIGURE 2.1. Bootstrap percolation dynamics with $\tau = 4$ and initial configuration of 70% probability of populating a cell over a 100×100 lattice with Moore neighborhood and periodic boundary. From left-to-right frames 1, 4, 16 and 64 are presented. Notice that distinct clumps of cells appear to form by iteration 64. The these clumps correspond to cells that are still in 0 state.

2.1.2. Game of Life (GOL). Any discussion of CA cannot be made without mention of the most famous example of a cellular automaton: John Conway's Game of Life (GOL). GOL is a CA that has very special self-replicating patterns. As seen in the introduction these patterns include gliders, but also "spaceships" and more. It is also said that GOL is capable of universal computation in that one can fit a Turing machine to produce any computable sequence [**9**]. GOL is aptly named so since its transition rules define whether or not a given cell becomes *born*, transition from 0 to 1; whether it lives, remains 1; dies, transitions from 1 to 0; or remains dead, stays 0 all according to its neighborhood population: over- or underpopoulated [**13**] [**14**]. Figure 2.2 shows example dynamics of GOL from a random initial density of 30% active cells—different moving groups of cells can be seen to move and become displaced through the lattice. The formula below formal denotes the phase transition function of GOL:

$$X_{t+1}(v) = \begin{cases} 1 \quad , \ \text{if } \sum_{u \in \mathcal{N}(v)} X_t(u) = 2 \text{ or } 3 \text{ and } X_t(v) = 1 \text{ or} \\ \qquad \sum_{u \in \mathcal{N}(v)} X_t(u) = 3 \text{ and } X_t(v) = 0 \\ 0 \quad , \ \text{otherwise} \end{cases} .$$

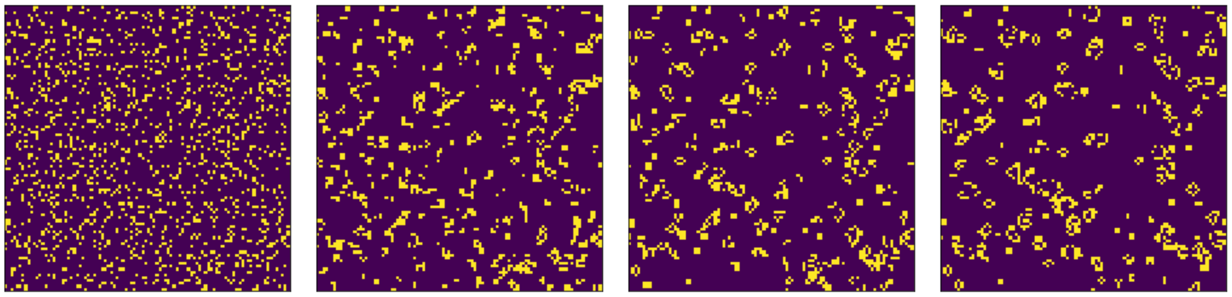

FIGURE 2.2. Game of life dynamics with starting initial configuration corresponding to 30% probability of populating a cell over a 100 × 100 lattice with Moore neighborhood and periodic boundary. From left-to-right frames 1, 2, 4 and 8 are shown. Some groups of cells seem appear to persist in time, but may disappear in time.

2.1.3. Cyclic Cellular Automaton (CCA). Moving on from binary CA such as BSP and GOL, cyclic cellular automaton (CCA) is a cellular automaton that can represent more than two discrete states, or κ states. Originally developed by David Griffeath, CCA in particular is interesting in that it can form spiral patterns that resemble the Belousov-Zhabotinsky reaction. Almost surely, a system of CCA dynamics on a lattice will enter a infinitely periodic cycle of states for each node on a lattice [**15**] [**16**]. Figure 2.3 shows CCA dynamics where we can clearly see example spirals forming on the lattice, again with periodic boundary. The formal definition for the CCA transition function is defined as follows:

$$X_{t+1}(v) = \begin{cases} 0, \quad \text{if } \exists X_t(u) = 0 \text{ for } u \in \mathcal{N}(v) \text{ and } X_t(v) = \kappa \\ X_t(v) + 1, \quad \text{if } \exists X_t(u) = (X_t(v) + 1) \text{ for } u \in \mathcal{N}(v) \text{ and } X_t(v) \neq \kappa \end{cases} ,$$

2.1.4. Greenberg-Hastings Model (GHM). The Greenberg-Hastings model (GHM) is conventionally discussed as a three state two-dimensional cellular automaton. Much of the original

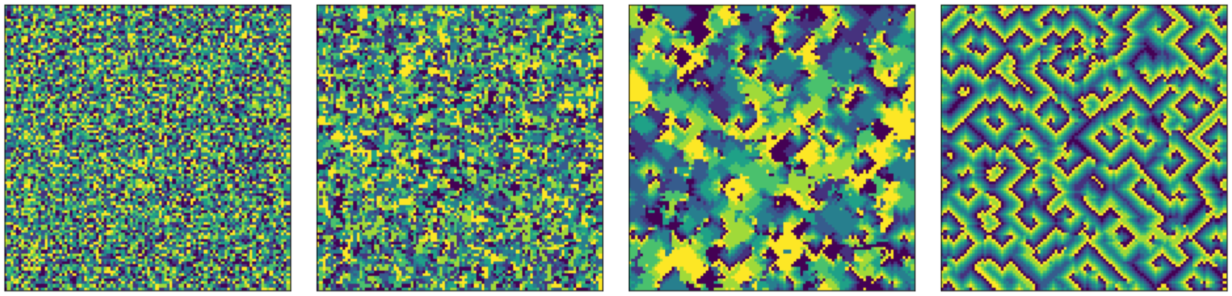

FIGURE 2.3. Cyclic cellular automata dynamics with $\kappa = 8$ starting from uniformly random initial configuration on a 100×100 lattice with von Neumann neighborhood. From left-to-right frames 1, 4, 16 and 64 are shown. An obvious pattern is maze-like structure throughout the lattice by time $t = 64$.

CA literature around GHM is peratins to models of "excitable media" for which GHM also simulates; conventional GHM models have three states: resting, excited and refractory. GHM is similar to CCA with respect to the inherent simplicity of its definition [**17**] [**18**] [**19**]. Spiral-like waves form at certain sites on a lattice similar to CCA. Figure 2.4 shows example dynamics for GHM when $\kappa = 8$, and below we provide a formal definition for its transition function:

$$X_{t+1}(v) = \begin{cases} (X_t(v) + 1) \mod \kappa & , \text{ if } X_t(v) \neq 0 \text{ or } \exists u \in \mathcal{N}(v) \text{ s.t. } X_t(u) = 1 \\ X_t(v) & , \text{ otherwise} \end{cases}.$$

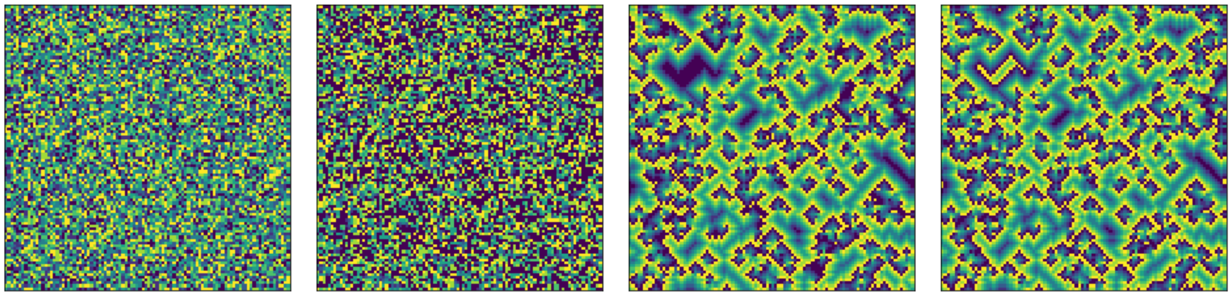

FIGURE 2.4. Greenberg-Hastings model dynamics with $\kappa = 8$ starting from uniformly random initial configuration on a 100×100 lattice with von Neumann neighborhood. From left-to-right frame 1, 4, 16 and 64 are shown. Intermediate dynamics at iteration 4 appear similar to CCA, but individual "islands" seem to pervade the lattice by iteration 64.

2.1.5. Firefly Cellular Automata (FCA). CCA and GHM models are similar in the sense that their transition functions are conditional on relative differences of states between neighbors (i.e., whether certain neighbors are in a certain state, or if the difference in states is less than another). The Firefly Cellular Automaton (FCA) presents a more complex condition for transition. Developed by Hanbaek Lyu, the inspiration for FCA comes directly from the phenomenon of the synchronization of blinking firefly lights. FCA are predominantly understood further as an of a *pulse coupled oscillator* where there is a specially designated *blinking color*, $b(\kappa) = \lfloor \frac{\kappa-1}{2} \rfloor$. FCA have been studied in many contexts combinatorially, probabilistically, and in empirically with statistics/machine learning techniques [**20**] [**21**] [**22**]. Figure 2.5 shows example dynamics, where we see from random uniform initialization, large square wave patterns begin to form, which is significantly distinct from CCA and GHM with more localized patterns—these cellular automata will not exhibit this behavior even if $\kappa = 4$. Below is the formal definition of the transition function for FCA:

$$X_{t+1}(v) = \begin{cases} X_t(v), \text{ if } X_t(v) > b(\kappa) \text{ and } \exists u \in N(v) \text{ s.t. } u = b(\kappa) \\ (X_t(v) + 1) \mod \kappa, \text{ otherwise} \end{cases},$$

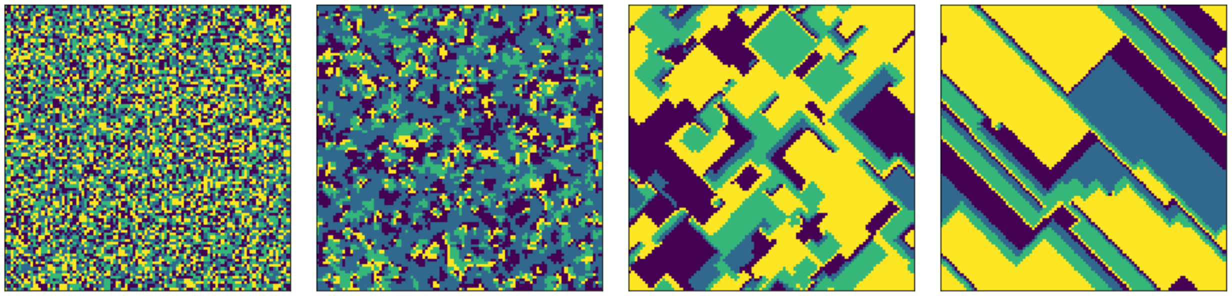

FIGURE 2.5. Firefly cellular automaton dynamics with $\kappa = 4$ starting from uniformly random initial configuration on a 100×100 lattice with von Neumann neighborhood. From left-to-right frame 1, 10, 100 and 1000 are shown. Significantly large groups of colors are present.

2.2. Graph Convolutional Networks

In this section we briefly discuss background on artificial neural networks and their variations. In particular, we describe convolutional neural networks—their inadequacies with respect to modeling CA rules—and the recent development of graph neural networks (GNNs), which is the core component in the process of producing these *neural profiles* of CA rules. We emphasize that learning the lookup table of a CA rule is a nontrivial task, especially since there have been no successful attempts. Some have used convolutional neural networks (CNNs) to learn Conway's Game of Life, but with little success [**23**]. In this section we discuss various neural network architectures and their pitfalls, building towards the graph convolutional network (GCN) model for the CA introduced in the previous section.

2.2.1. Artificial Neural Networks (NNs). Inspired by real-world neurons, artificial neural networks (NNs) work by stacking layers of artificial neurons operating as computational units performing arithmetic on input data from subsequent layers. For problems such as multiclass classification one may be given an input data matrix $X \in \mathbb{R}^{n\times m}$ of n observations and m features. An example ANN can be designed with an input layer, one hidden layer of K nodes, and an output layer of P nodes [**24**]. Additionally, what makes ANNs unique from just matrix multiplication and linear models is that at each layer, a nonlinearity is introduced with an *activation function*, $\sigma(\cdot)$, that is applied entrywise to each entry of a transformed data point through each layer. The main requirements for this activation is that is continuous and monotonic. This process of moving data sequentially through these layers of neurons corresponds precisely to what is known as *forward propagation.*

As an example, we briefly explain forward propagation of an ANN for data with m features and P dimensional output corresponding to P class classification (also known as *one-hot encoding* of a categorical variable); the following describes a single forward pass for a single hidden layer ANN:

(1) Input layer to hidden layer: A single data point, $\mathbf{x}$, has dimensions $1 \times m$, or $\mathbf{x} \in \mathbb{R}^{1\times m}$ and undergo the operation

$$\mathbf{x}^{(0)} = \sigma\left(\mathbf{x}\mathbf{W}_0 + \mathbf{b}_0\right),$$

with $\mathbf{W}_0 \in \mathbb{R}^{m\times K}$ and $\mathbf{b}_0 \in \mathbb{R}^{1\times K}$. Also, the activation function $\sigma(\cdot)$ will be applied entrywise to $\mathbf{x}^{(0)}$, which is consistently a row column as intended.

(2) Hidden layer to output layer: Again, between the input and hidden layers we have the first transformation of $\mathbf{x}$, $\mathbf{x}^{(0)} \in \mathbb{R}^{1\times K}$, passing through hidden layer to the output layer

as

$$\mathbf{x}^{(1)} = \sigma\left(\mathbf{x}^{(0)}\mathbf{W}_1 + \mathbf{b}_1\right),$$

with $\mathbf{W}_1 \in \mathbb{R}^{K\times P}$ and $\mathbf{b}_1 \in \mathbb{R}^{1\times P}$. Once again, the activation function σ will be applied entrywise to $\mathbf{x}^{(0)}$, a final row column of desired shape, $\mathbf{x}^{(1)} \in \mathbb{R}^{1\times P}$. After optimal training and tuning, one hopes that for $\hat{y} := x^{(1)}$, $\hat{y} \approx \mathbf{y}$, where the estimated $\hat{y}$ output is approximately true output, $\mathbf{y}$, corresponding to input $\mathbf{x}$.

Note that there will always be an input and output layer. In this example, there is a third "hidden layer," but there are a total of two weight matrices and two bias vectors. Therefore, there is necessarily $L-1$ affine transformation, weight matrices where L is the number of layers in the NN. Above, we have an input layer, hidden layer and output layer, or $L = 3$, and we have input-to-hidden and hidden-to-output weight matrices, or two weight matrices. Fitting an ANN model—or more generally, having an neural network *learn*—involves utilizing algorithms in the complementary step of neural network learning known as *back propagation.* Back propagation is essentially of minimizing a designated loss function, or simply optimization [**24**]. Stochastic gradient descent (SGD) is an example of algorithm utilized in back propagation where random batches of the full dataset are passed during forward propagation and back propagation at each iteration as opposed to performing forward and backward propagation on the whole dataset.

Formally, stochastic gradient descent does not refer to a direct randomness in of the gradient descent procedure, but the in randomness that is incorporated when computing each gradient step through batching. Backpropagation with SGD is usually incorporated by randomly batching the training dataset into batches of size $\tilde{N}$ at each iteration (e.g., at each iteration for 110 training data points with batch size 30, randomly sample without replacement 30 data points for the first three batches and 20 data points for the last batch for the remaining data points). Then for each of these batches SGD perform both forward propagation and backpropagation to learn the weights and biases. The beyond random batching of the training dataset, the process of backpropagation utilizing a given optimization involves solving matrix derivatives of the NN weights and biases with respect to a *loss function* which the NN aims to minimize.

In our example, cross entropy loss is a common choice for a loss function in multiclass classification where the labels are one-hot encoded (i.e., numeric label k of P classes corresponds to

P-dimensional standard basis vector, $\mathbf{e}_k$):

$$\ell(\mathbf{W},\mathbf{b}) = \frac{1}{\tilde{N}_k}\sum_{n=1}^{\tilde{N}_k} \mathrm{CE}_n \text{ with } \mathrm{CE}_n = -\mathbf{x}_n^{(1)} \cdot \log\left(\mathbf{x}_n^{(1)}\right),$$

where $\mathbf{x}_n^{(1)}$ corresponds to the final output of data point $\mathbf{x}_n$ in our forward propagation step and we apply $\log(\cdot)$ entrywise to vector $\mathbf{x}_n^{(1)}$; $\tilde{N}_k$ denotes the sample size of the kth random batch of the training dataset. Implicitly, in the final expression, we can see that

$$\mathrm{CE}_n = -\mathbf{x}_n^{(1)} \cdot \log\left(\mathbf{x}_n^{(1)}\right) = -\sigma\left(\sigma\left(\mathbf{x}_n W_0 + b_0\right) W_1 + b_1\right) \cdot \log\left(\sigma\left(\sigma\left(\mathbf{x}_n W_0 + b_0\right) W_1 + b_1\right)\right),$$

and, once again, we would perform a complicated series of steps to compute the gradients and derivatives with respect to $\mathbf{W}_0$, $\mathbf{W}_1$, $\mathbf{b}_1$ and $\mathbf{b}_0$ (luckily, deep learning frameworks such as PyTorch and Tensorflow exist to handle complex backpropagation computations effortlessly).

Another common choice that is utilized during the training of neural networks is to include neuron *dropout* at each iteration. Dropout refers randomly converting the output of a neuron at a given hidden layer to zero on the forward pass according to some fixed probability p—the idea behind such dropout being that it will allow the ANN to more carefully and slowly traverse the manifold of the chosen ANN loss function. Different choices of optimization and loss function corresponding to dataset size and type (multiclass versus scalar continuous), as well as the number of hidden layers and neurons can be tuned according to various heuristics and performance metrics, which is to say that there are a seemingly endless number of possible choices to be made when designing a neural network to learn a dataset. Another way of thinking about the effect of dropout is that the $\sigma(\cdot)$ activation function behaves exactly like a random function with probability p outputting a zero value. Both dropout and SGD have the same effect of ensuring that the loss function is able to escape local minima through random variation of gradient step magnitude and direction.

2.2.2. Convolutional Neural Networks (CNNs). Standard ANNs have plenty of applications and are likely sufficient for simple toy datasets where all the data can be accurately embedded in a standard design matrix, X, where the number of rows corresponds to the number of observations and the number of columns, m, corresponds to the number of features for each data point for which there may be n data points. A natural solution to learning these lookup tables on lattices is to imagine these neighborhoods as 3×3 images. Methods in image classification can handle this Euclidean representation, but are not necessarily optimal. If one has a dataset of multiple images corresponding to N different classes, it may be difficult to engineer features that efficiently and effectively distill qualities of an image to be learnable in a Euclidean representation (i.e., 3×3 images cannot be engineered into features and such features will not be consistent between different CAs). An image may be converted from a $K \times K \times 3$ tensor, corresponding to image dimensions ($k \times k$) and RGB values (size 3), into features such as pixel density of RGB values, relative pixel hue, saturation and brightness—the image dataset may be engineered into a very simple and very inadequate representation of 6 features.

A solution to this inadequate representation can be to utilize convolutional neural networks (CNNs). CNNs can directly learn from an image representation without need for engineering features. In addition to the standard neural network architecture of different layers, CNNs have the addition of convolutional layers that produce convolved images from the original input image essentially pooling and aggregating local image features into more coarse representations through each layer in its architecture [**25**] [**26**]. Kernels are the main component of CNNs and can be tuned with choice of padding, stride, number of kernel filters, and kernel pooling type—it is precisely named *kernel* as understood in Fourier analysis for similar applications of convolutions operators on L^2-integrable functions. (More details on CNNs images can be found here [**27**].)

2.2.3. Graph Convolutional Networks (GCNs). Graph-based data have become increasingly popular due to their ability to accurately and precisely model real-world networks (e.g., social networks and epidemiological models). Graph neural networks (GNNs) is a catch-all name for neural networks designed to handle problems and tasks related to graph structured data. GNNs have numerous applications from graph classification over datasets of graphs to node classification on individual graphs; GNNs can also be used to determine whether edges exists between nodes—edge detection—and more [**28**] [**29**]. Graph convolutional networks (GCNs) are a variation of GNNs that are utilized for node classification problems—this will be the main tool for developing our models of cellular automata in this thesis.

Here we briefly discuss forward propagation for single-layer GCN on a single node with binary output (node classification)—we will suppose that our neural network has a single hidden layer of K neurons. In particular, consider the following tree graph:

$$G = (V, E) = (\{\mathtt{A}, \mathtt{B}, \mathtt{C}, \mathtt{D}, \mathtt{E}\}, \{\{\mathtt{A}, \mathtt{B}\}, \{\mathtt{A}, \mathtt{C}\}, \{\mathtt{A}, \mathtt{D}\}, \{\mathtt{A}, \mathtt{E}\}\}),$$

and let `A` be the *target node*. Additionally, suppose each node has a single scalar feature associated with it and that edges are directionless and unweighted, constant weight, 1. Also, suppose we are performing P class classification such that we have again one-hot encoded the labels as P-dimensional vectors. We now discuss the steps of GCN forward propagation below:

(1) Input layer to hidden layer: For each node, k, in the vertex set, or node set, V, we first compute

$$\mathbf{h}_k^{(1)} = \mathbf{W}^{(1)}\mathbf{x}_k^{(0)} + \mathbf{b}^{(1)}, \ \ \forall k \in V,$$

corresponding to affine feature transformations where $\mathbf{W}^{(1)} \in \mathbb{R}^{K\times 1}, \mathbf{b}^{(1)} \in \mathbb{R}^{K}$ since $\mathbf{x}_k^{(0)}$—denoting features of node k—is single dimensional. Then we compute the following:

$$\mathbf{x}_i^{(1)} = \sigma\left(\mathbf{h}_i^{(1)} + \sum_{j\in\mathcal{N}(i)} w_{ij}\mathbf{h}_j^{(1)}\right)$$

where we are perform feature aggregation of the neighbors of node i by taking a linear combination of the $\mathbf{h}_k^{(1)}$ by w_{ij}, which are again just constant edge weight, 1. Finally, we perform a nonlinear entrywise activation to get $\mathbf{x}_i^{(1)}$ with $\sigma(\cdot)$, giving us a K-dimensional vector.

(2) Hidden layer to output layer: This step is essentially identical as the last. Consider the following:

$$\mathbf{h}_k^{(2)} = \mathbf{W}^{(2)}\mathbf{x}_k^{(1)} + \mathbf{b}^{(2)}, \ \ \forall k \in V,$$

$$\mathbf{x}_i^{(2)} = \sigma\left(\mathbf{h}_i^{(2)} + \sum_{j\in\mathcal{N}(i)} w_{ij}\mathbf{h}_j^{(2)}\right),$$

where $\mathbf{W}^{(2)} \in \mathbb{R}^{P\times K}$ and $\mathbf{b}^{(2)} \in \mathbb{R}^{P}$; we then aggregate these features and apply the activation function giving us the desired two-dimensional output.

An important thing to note is that in this single hidden layer GCN, if we apply forward propagation to target node `A`, we are aggregating information of its neighbors from $\mathbf{x}_j^{(2)}$ for all $j \in \mathcal{N}(\texttt{A})$, which are aggregated from their neighbors, which is node `A` itself. In other words, for a uniform 1-depth tree, adding layers additional layers induces a sort of recursive aggregation of feature information towards the target root node, `A`, and its neighborhood, $\mathcal{N}(\texttt{A})$.

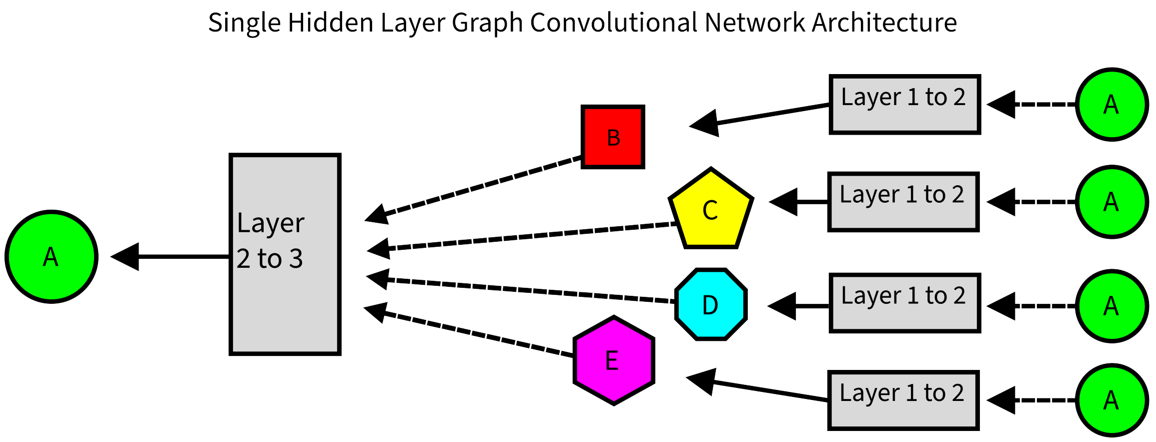

FIGURE 2.6. Architecture of single hidden layer GCN. Applying a GCN to the root node of a uniform, 1-depth tree graph feeds information from the neighbors of neighbors of `A`, which is just the root node itself, `A`. Effectively, a single hidden layer neural network has a recursive information structure where information about the target node is passed pairwise to its neighbors, transformed and aggregated, and then passed back to the target node for transformation and aggregation once more—a feedback loop induced by this strict tree structure embedding of the local neighborhood.

Although forward propagation can be done for every node on a graph, with respect to neural network training, we can specify and designate training nodes such that only certain nodes nodes from the original node set are computed in the loss function during forward propagation. For example, since GCNs are node classifiers, we can train our GCNs on just the root node of a dataset of multiple tree graphs and specifically avoid training the GCN on child nodes or "leaves." Also, back propagation works exactly the same for GCNs as ANNs, and cross entropy loss would again be a natural choice for loss function in the example above (i.e., various choices of optimization technique, loss function and dropout parameters, etc.).

Two main characteristics highlight the remarkable utility of GCNs for graph-based data with special application in node classification:

(1) (***Permutation Equivariance***) Permutation equivariance defines the property that the features of any target node and its neighbors are combined in a way that is independent of any orientation or relative displacement around the target node. For example, a CNN does not aggregate information in an equivariant manner since the orientation and relative displacement of pixels values in images are important in discerning certain subjects and patterns—permutation equivariant aggregation instead combines local information such that this orientation does not matter.

(2) (***Message Passing***) Message passing in GNNs is very significant for node classification problems since it allows a neural network to precisely extract and learn information for a given node based on its directly connected local neighborhood. We can construct a tree embedding of states as described above for a von Neumann neighborhood, such that we have realized a mode of representation that uniquely establishes a correspondence between a target node and its neighborhood.

As distinct as GCNs appear to be from CNNs, they preserve the desired behavior of aggregating local neighborhood information such as a lookup table mapping a neighborhood, state-space configuration to the center target node's update, but with the desired addition of permutation equivariance and message passing. In this sense, GCNs appear to be the perfect model for totalistic CA, state-transition functions.

2.3. Modeling Cellular Automata Rules

With the introduction of different cellular automata models and rules, as well as graph convolutional neural networks, this section discusses the process for learning lookup tables and developing cellular automata neural profiles.

2.3.1. Data Representation. The key objective of this study is to learn and represent cellular automata rules operating on lattices with Moore and von Neumann neighborhoods. The main problem for learning lookup tables according to some model is developing a way of doing so in a way that avoids issues in lack of permutation equivariance as discussed in the previous section. The corresponding dataset for any particular cellular automata (CA) rule, $X_t(v)$, is relatively easy to specify—the main problem is the actual representation of the data for modeling. For example, on Moore neighborhoods all that needs to be done is to produce all possible 3×3 lattice combinations of κ states, and then apply the CA rule over each combination to produce labels, of which there will be at most κ distinct outputs (for the purposes of machine learning, again, we would one-hot encode these states as vectors). One could suggest flattening all 3×3 configurations as 9×1 vectors with output corresponding to the state of the center node at the next timestep, $X_{t+1}(v)$, for a Euclidean representation of the lookup table data.

The center node can be fixed at the first column of the data matrix and all configurations can be upsampled such that all possible permutations in flattened configurations are produced, and we can learn from this naïve Euclidean representation. Yet, this representation has major flaws since it implies that the resulting data matrix, X, from this representation has columns corresponding to distinct features with no dependence to their relative displacement from the target update node (e.g., we throw away information on how the neighbors are connected to the target node).

Naturally, a standard neural network cannot effectively learn the mapping from 3×3 neighborhood configurations to $(t + 1)$ state transitions. We can then try learning the 3×3 configurations as images as part of an image classification setting. The problem of neighborhood node displacement is solved since the exact neighborhood configurations can be given as individual images or data points. But CNNs are sensitive to pixel permutation and displacement. In other words, in a dataset where each "image" has exactly 9 "pixels" a CNN would be very sensitive in its learning procedure to the permutation and contents of each node. This problem is further exacerbated if one attempts to upsample and produce all permutations since the the kernel layers aggregating pixels are not permutation equivariant—each layer in a CNN will learn isomorphic configurations distinctly. Consider Figure 2.7, a representative example of a 3×3 neighborhood configuration that one could see on a CA with five unique states. A totalistic cellular automaton transition rule will treat these neighborhoods precisely in a permutation equivariant manner; the counts of the neighboring vertex states are exactly the same, and their orientation/permutation does not affect the next state of the center, target node; however, a CNN will not.

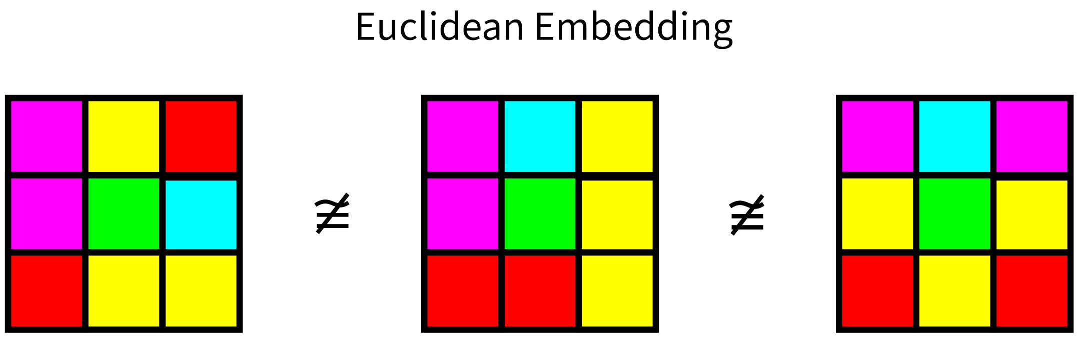

FIGURE 2.7. Example configurations of local neighborhood on a lattice. For a CA transition function, the target update node would correspond to center green cell. Clearly the frequencies of colors or states around the green node is the same, yet the configurations are not equivalent in this standard Euclidean embedding.

Furthermore, since a CNN convolves areas over entire images, it may try to learn an association in this 3×3 configuration that has nothing to do with the CA transition rule. For example, again, in Figure 2.7, it may try to learn a pattern of corner upper left corners of size 2×2 where we have two adjacent pink cells correspond to a particular label. Yet, in standard and commonly studied CA such neighborhood patterns are not relevant to the typical totalistic CA. The main interaction of information in a neighborhood around a target node is the relationship of the neighboring nodes to the target node, not the interaction of the nodes in the neighborhood themselves.

This leads us to an ideal and more elegant representation of these 3×3 configurations as 1-depth trees of degree 4 and 8 for von Neumann and Moore neighborhoods, respectively. Defining these configurations as trees allows us to deliberately utilize message passing precisely between nodes from the neighboring nodes to the target node. Figure 2.8 shows this representation in action: we can see that the issue of position dependent information from standard image-like representations is entirely avoided when we define all Moore or von Neumann neighborhood configurations as trees that directly pass on the conditions of state transition from each neighbor directly to the target node.

Furthermore, this representation also allows us to produce—and learn from—datasets that efficiently represent local neighborhoods for a given target node without any confusion of neighboring node orientation. In particular, this mode of data allows us to utilize the permutation equivariance

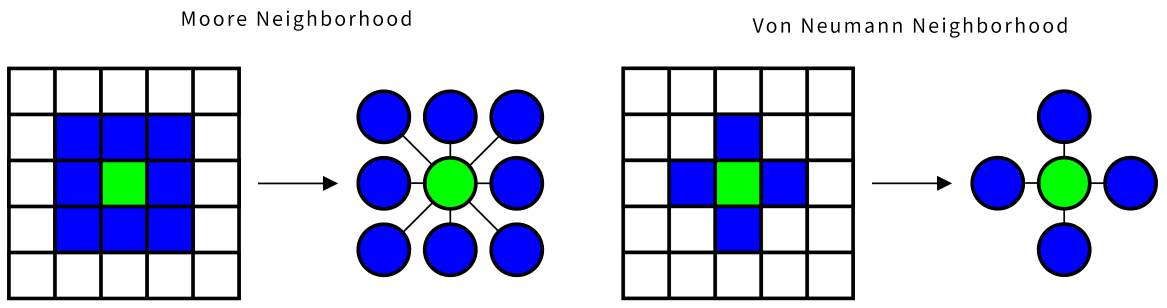

FIGURE 2.8. Representing lattice neighborhoods as trees. The center node of these neighborhoods serve as the root of these trees, and all the neighbors are deliberately paired straight to the target node. The issue of any mixed information between neighboring, non-target nodes is avoided with this graph embedding of Moore and von Neumann neighborhood.

property that is intrinsic to standard totalistic cellular automata models studied in the literature—and GCNs are precisely and exactly the optimal machine learning model to perfectly learn these abstract lookup tables as exampled in Section 2.1. Figure 2.9 provides an example von Neumann neighborhood configurations as tree representations that are isomorphic to each other such that they will be aggregated in a permutation equivariant manner with respect to a GCN.

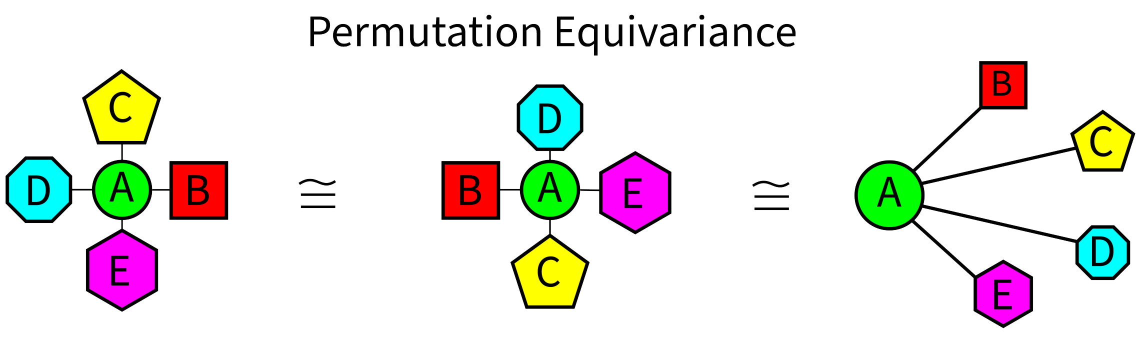

FIGURE 2.9. Example of local von Neumman neighborhood of nodes. Even with different orientations of these graphs rendered physically in two-dimensions as seen in this figure, these graphs are precisely isomorphic having the exact same node features and labels. The key point of the third graph is to show that for a GCN, it matters not so much that the surrounding nodes make up the neighborhood of node `A` as much as they are directly connected to node `A`—this vital relationship is what is captured with the message passing property of GCNs.

2.3.2. GCN Modeling. As formulated and presented, a GCN and the CA introduced in this study are not at all limited to lattices. These CA can operate on random graphs with complex and irregular node degree distributions. However, in limiting our study to lookup tables corresponding to lattices with Moore and von Neumann neighborhoods, the possible complexity of models in this lattice setting is significantly reduced which is important for feasible analysis. Again for the sake of feasible analysis, while we study five different cellular automata models, we are limited to CA models of two and three states.

In this study, nearly all the cellular lookup tables for von Neumann and Moore neighborhoods are perfectly learned using only two hidden GCN layers. In particular, with the corresponding transition rules defined in Section 2.1, what's nice about limiting the scope of possible neighborhoods for CA like GHM and CCA is that these CA all have the exact same GCN architecture and model (i.e., a single overparameterized model for different CA lookup tables). For example, we simulate CA lookup tables for GOL and BSP over Moore neighborhoods such that these CA will have the same model architecture. For GHM and CCA, these models over three states on von Neumann neighborhoods will also have the architecture as seen in Figure 2.10. This architecture corresponds to precisely how configurations of vertices in Figure 2.9 pass information to the target node through each hidden layer. (Note that in 2.10 that although there are two hidden layers, there are three corresponding weight matrices and three bias vectors as can be induced from our single hidden layer example from the previous section.)

The only cellular automata model that could not be learned two hidden layers from our simulations was FCA. GCN models with two hidden layers and varying number of hidden neurons failed to exactly learn three-state FCA. Only after adding a third hidden layer were we able to perfectly learn the FCA lookup table. Furthermore, between different instantiations of three hidden layers GCNs for FCA, the rules that it failed to learn were also random with no consistent neighborhood failing to be learned. For the remainder of the thesis we will use *four layer GCN* to refer to a GCN with two hidden layers (Note that in the three hidden layer case, we end up with four associated weight matrix and bias vector pairs corresponding to five total layers including the input and output layers.)

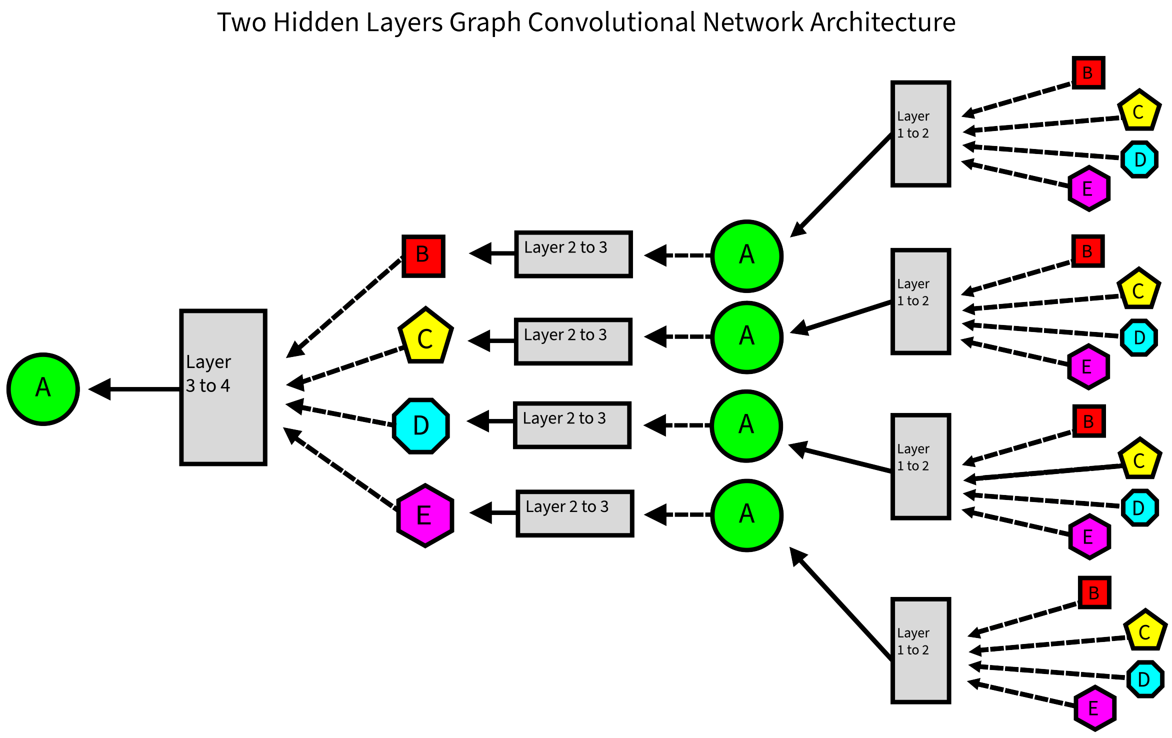

Figure 2.10. Architecture of two hidden layers GCN. This single directional architecture signifies features or states of the neighborhood of nodes passing information and being aggregated through the each layer to the target node that will be updated. In this diagram "Layer 1" corresponds to the input layer and the "Layer 4" corresponds to the output layer. Layers 2 and 3 refer to the hidden layers. Note that there are three corresponding weight matrices and bias vectors in this GCN with four total layers (input-to-hidden, hidden-to-hidden, and hidden-to-output).

The only missing piece behind this model of cellular automata rules is the utilization of multiple samples of GCNs to represent a *consistent numerical characterization* of transition rules. Namely, when one fits any neural network, the training process iteratively updates and learns the weights and biases of the neural network. These weights are randomly initialized and optimize through back propagation as detailed in the previous section. While it is sufficient to model a rule with just one learned/fitted GCN model, due to the randomness and sensitivity of the initialization of GCN weights and biases, numerically characterizing a CA lookup table with only one fitted GCN is insufficient. The requirement of multiple samples of GCNs perfectly fitting the lookup table of various CA is required to develop a sampling distribution of weights matrices of these cellular automata rules. These repeated samples—starting from different random initializations—all learning the exact cellular automaton lookup table is what consists of the *neural profile of the cellular automaton.*

2.3.3. Data and Model Specifications. In this subsection we specify some characteristics of the datasets generated from the five cellular automata described earlier in this chapter. Importantly, since neural networks, and machine learning models in general, are sensitive to the balance of labels for classification we upsample data points according to their least common multiple. For example, for binary distributions, if there are 12 data points of label 0 and 33 data points of label 1 according to binary classification, we upsample each (neighborhood, update state) pair by 11 and 4 for labels 0 and 1, respectively—this creates a uniform split of labels in our dataset ensuring that no single label is overly predicted. This same process is applied for cellular automata with three states.

- Bootstrap Percolation (BSP): For BSP, we generate all bootstrap graph datasets and configurations for $\tau = 0, 1, 2, 3, 4, 5, 6, 7, 8$. BSP configurations are generated over 1-depth trees of degree 8 (Moore neighborhood). The number of data points for each threshold level is shown in Table 2.1. Below we provide specifications on the datasets generated.

Table 2.1. Number of data points per label after upsampling for BSP on each possible τ value. Note that there BSP

Threshold (τ)	0	1	2	3	4	5	6	7
Unbalanced Labels (0:1)	1:17	2:16	3:15	4:14	5:13	6:12	7:11	8:10
Balanced Labels (0/1)	17	16	15	28	65	12	77	40

- Game of Life (GOL): For GOL configurations are generated over 1-depth trees of degree 8 (Moore neighborhoods). There are only three configurations resulting in $X_{t+1}(v)$ state becoming 1, and 15 configurations corresponding to a 0-label. After upsampling the data points for balanced labels, we end up with 15 data points each for 0-labels and 1-labels.
- Cyclic Cellular Automata (CCA): For CCA, we consider $\kappa = 3$ possible states generated on von Neumann neighborhoods. In particular, there are 15 data points for each state 0, 1 and 2. No balancing was needed.

- Greenberg-Hastings Model (GHM): For GHM, before balancing there are 20, 10 and 15 data points for labels 0, 1 and 2, respectively. After upsampling for balance, each state has 60 data points. We produce configurations over von Neumann neighborhoods.
- Firefly Cellular Automata (FCA): For FCA, there are 5, 15 and 25 data points corresponding to labels 0, 1 and 2. After labeling balancing we use 75 data points for each state in training. We produce configurations over von Neumann neighborhoods.

The balanced sets are used for neural network training, and the unbalanced sets are used to validate training performance. Again, since we want to exactly model the lookup table with GNNs, we make sure that the validation performance, which is just validation accuracy on the training set itself, consisting of all neighborhood configurations with update labels, is met at 100% accuracy.

The stopping condition used for training is training at 2,500 max iterations, or until the validation accuracy is met at 100%. Additionally, we have 50% neural network node dropout for regularizing purposes—50% is a heuristic choice with most dropout rates being common among machine learning practioners. We also use the Adam optimizer with a learning rate of $\alpha = 0.001$ and weight decay of `5e-4`, and the choice for our activation function is the rectified linear unit (ReLU), which is simply defined as $\sigma(x) = \max\{0, x\}$ [**30**] [**31**]. Since we have output labels of 0/1 and 0/1/2 for the above cellular automata, we use cross entropy loss as our loss function, and again, our classes are one-hot encoded. The random intialization of the weights are generally dependent on the choice of activation function, which in this study was the ReLU activation, we utilized He intialization [**32**]. (All of the code used to generate the datasets as well as the implementation of GCNs on datasets described this study are available at `https://github.com/richpaulyim/neural-profiles` . All code was implemented in with PyTorch and in particular Torch Geometric and neural networks models were run on a `CUDA` enabled RTX card `Python3.10` [**33**].)

CHAPTER 3

Spectral Analysis of Cellular Automata Neural Profiles

3.1. Damage Spreading

Before formally analyzing neural profiles of various CA we first introduce one of many possible ways to characterize CA stability: damage spread. One of the main purposes of understanding CA stability in our study is to provide a qualitative characterization that can be linked to neural profiles. Damage spread measures stability by perturbing an initial configuration and observing an accumulation or dissipation of *damage* between the original configuration as the dynamics of the cellular automaton are simulated through time. Other methods such as Lyapunov exponents which are common in conventional dynamical systems theory have discrete extensions for CA, but this characterization is too rigorous such that it has no natural extension to CA in two dimensions [**34**]. Damage spread is a very flexible tool for characterizing stability of CA on all domains and over any κ states.

Damage spread has a very general definition, which we formally define for graphs: given a configuration of κ states over an N-node graph, G, and a uniform random perturbation of κ states on an n-node, single-component induced subgraph of G, F, where $n < N$, damage spread at time T is defined as the proportion of nodes between graph G and F with differing states. The following subsections will look at damage spread of different CA.

3.1.1. Two-state Cellular Automata Damage Spread. For BSP, as well as all the other CA in this section, we consider 60×60 lattices with periodic boundary to characterize damage spreading properties—for BSP and GOL we are, again, in a Moore neighborhood setting. Due to the randomness in these perturbations we characterize damage spread over 30 samples of BSP damage spread dynamics. (*Damage spread dynamics* for any given CA will refer to the state-space dynamics of the difference in states over a fixed domain as opposed to a scalar measure of the proportion of damaged nodes.) In particular, we randomly perturb a 30×30 sublattice for each random initialization on the larger 60×60 lattice. In Figure 3.1 corresponding to BSP ($\tau = 4$), we have *average damage spread dynamics* where we superimpose 30 damage spread dynamics simulated from $t = 0$ to $t = 32$ and compute the average damage at each node for each time slice.

To observe the damage spread on these lattices by the definition introduced earlier, we refer to Figure 3.2, which shows the sample average damage spread at each iteration from $t = 0$ to $t = 50$ for $\tau = 0, 1, \dots, 7$. We denote the average damage spread of a CA on a graph, G, at time t as $\mathbb{E}[\Delta(G_{CA})_t]$. For $\tau = 0, 1, 2$ and 3 we have that the damage goes to zero and for $\tau = 4$ the damage is nearly zero, but still positive. For $\tau \geq 5$, the damage spread appears to be bounded below and remain static for all t. We can clearly see that between $\tau = 4$ and $\tau = 5$ the damage spread stability goes through a transition between strong stability—damage spread dissipating—and weak stability—damage spread stabilizing and becoming static. Furthermore, it is interesting to note that average damage spread for BSP is monotonic in time for both strong and weak stability cases (this makes sense since the states on a lattice for BSP *percolates* to a 1-state such that 1-states never become 0-states).

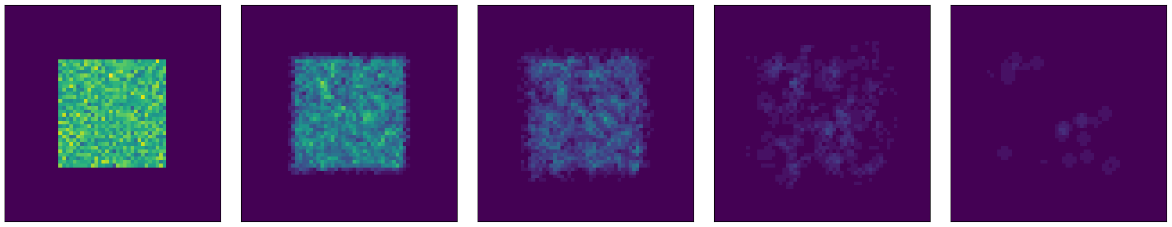

Figure 3.1. From left-to-right: average damage spread dynamics of BSP, $\tau = 4$, for frame $t = 0, 2, 4, 8, 16$. For 30 samples of damage spread simulations we see that the damage is sustained up until the last static frame for $t = 16$.

In addition various strong and weak damage spread stability properties shown by BSP, GOL similarly exhibits weak stability, but does so stochastically. Observing a few frames of the average damage spread dynamics of GOL in Figure 3.3, we can guess that the damage spread appears to continuously grow, but frame $t = 64$ compared to $t = 32$ shows lower density in average damage spread. This is precisely the case as the average damage spread decreases and weakly stabilizes after a sufficient number of iterations. Figure 3.4 shows this behavior: the damage spread appears to increase sharply as the damage spreads throughout the lattice, but over time it dissipates throughout the lattice stabilizing within a bounded range by $T = 1000$. To be precise, it stabilizes

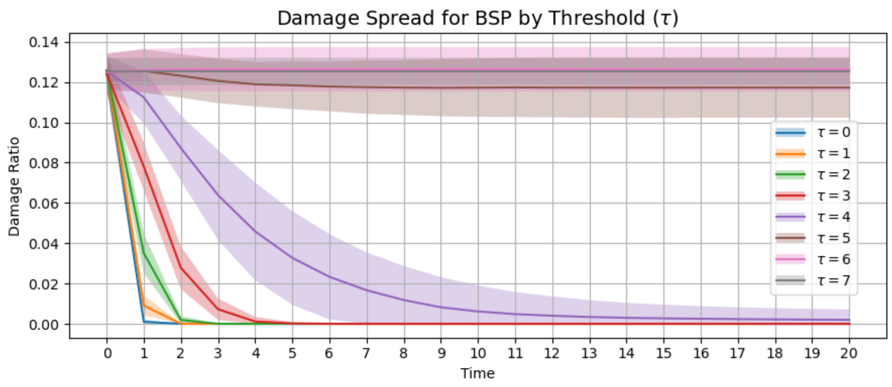

FIGURE 3.2. Average damage spread with 95% confidence bands for BSP by $\tau = 0, 1, \ldots, 7$ over 30 samples each. We can see that there the damage dies off to zero for $\tau = 0, 1, 2$ and 3 for 30 samples. For $\tau = 4$ the damage does not die off to zero for all 30 samples. For $\tau = 5$, the damage stabilizes and only slightly disappears; for $\tau > 5$, we see that there is little change from the initial damage.

to about the same damage spread density as expected if we ran only GOL initialized from a uniform random distribution.

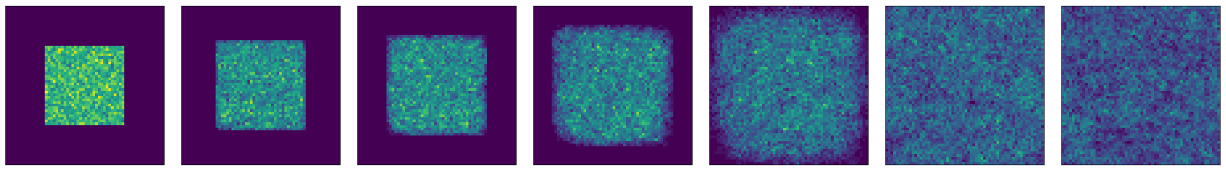

FIGURE 3.3. From left-to-right: average damage spread dynamics of GOL for frame $t = 0, 2, 4, 8, 16, 32, 64$ for 30 samples of GOL. In the case of GOL we see that the damage spreads monotonically in time with no sign of dissipation. Running the damage spread dynamics for GOL the damage spread appears to continue.

From these distinct two-state CA, we have observed two types of damage spread: strongly stable, $\limsup_{t\to\infty} \mathbb{E}[\Delta(G_{CA})_t] = 0$, and stable, $\limsup_{t\to\infty} \mathbb{E}[\Delta(G_{CA})_t] = c$ for $c > 0$. A distinction should be made however between the damage spread of BSP for $\tau > 4$ and GOL. Namely, if we compare Figure 3.1 and Figure 3.3 we observed the region of damage is dispersed throughout the entire lattice at a lower density for GOL, whereas, damage for BSP $\tau > 4$ will be confined within a small neighborhood of the initial site of damage.

3.1.2. Three-state Cellular Automata Damage Spread. We similarly observe damage spread for three-state CA introduced earlier (we again utilize a 60×60 lattice with periodic boundary

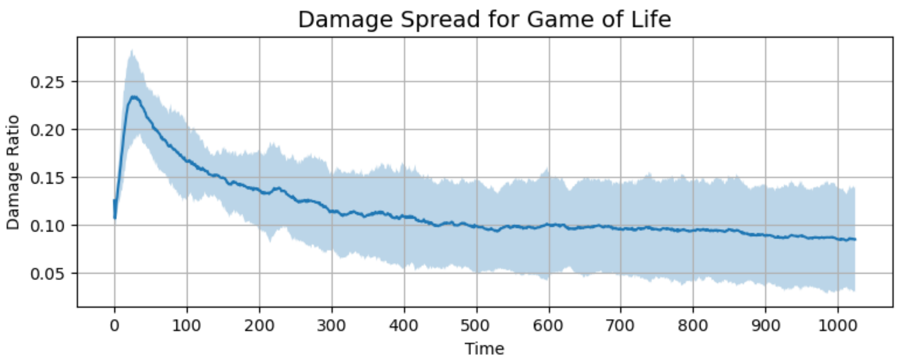

FIGURE 3.4. Average damage spread with 95% confidence bands for Game of Life for 30 samples. Damage spread increases just by the damage occupying the full lattice within the first 50 iterations, but quickly dissipates as the total damage spread density stabilizes to about 0.08. Note that unlike BSP, GOL does not stabilize monotonically in time.

perturbing a 30×30 region). Again, for these three-state CA we focus on lattices where the nodes are arranged in a von Neumann neighborhood. Figure 3.5 shows the average damage of the three different CA over 30 samples by time $t = 64$. Interestingly, these CA all share similar average damage spread densities. However, an important observation to make is that the average damage spread for FCA, unlike CCA and GHM, has strictly bounded oscillatory behavior—this can be seen in Figure 3.6.

With respect to the average damage spread over each iteration, we see in Figure 3.6 that the damage spread is generally weakly stable for CCA and GHM. We observe a third stability characteristic distinct from weak and strong stability: periodic stability, where the damage spread proportion oscillates for all time after a certain iteration. Other interesting points is that while the average damage spread is greater for CCA than GHM the relative difference in effect size is not significant.

3.2. Spectral Analysis of Hidden Layer Weight Matrices

We have specified the process for modeling CA lookup tables and producing neural profiles for various CA. We have also introduced a complementary extrinsic characterization of CA that is flexible to numerous types of CA over various domains with its very general, yet formal, definition. In this section we analyst these neural profiles. In particular, we focus on analyzing the hidden

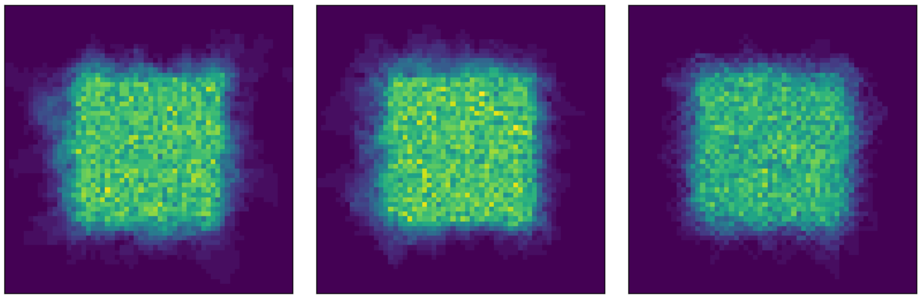

Figure 3.5. Average damage spread dynamics for CCA, GHM and FCA at time $t = 64$. The average damage spread dynamics is static by $t = 64$ for CCA and GHM; FCA damage spread oscillates within a bounded region.

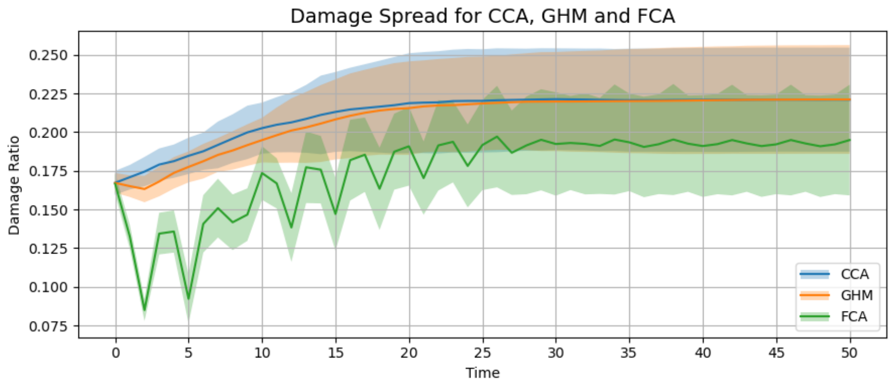

Figure 3.6. Average damage spread for CCA, GHM and FCA over three states ($\kappa = 3$) with 95% confidence bands. A distinct feature in the damage spread for FCA is that while the damage increases, like CCA and GHM, the average oscillates for all time.

layer weight matrices as one of many possible ways to analyze the data sampled from these neural profiles.

Since we manage to learn both two-state and three-state CA on lattices with the four layer GCN we can easily study any of the learned parameters associated with the multiple random samples of GCNs for each CA, namely the three pairs of weight matrices and bias vectors corresponding to affine transformations between layers after each activation function is applied. In this thesis we focus on studying the second matrix associated with transforming information between hidden layers, $\mathbf{W}^{(1)}$, which we will refer to as the *hidden matrix*, and denote as $\mathbf{H}_1$. We can also study the $\mathbf{W}^{(0)}$ and $\mathbf{W}^{(2)}$ matrices, but we focus here on the $\mathbf{H}_1$ matrices which are constructed as square matrices with significantly more variation and information compared to the the weight matrices between the input and hidden layer, and between hidden layer and output layer, which have dimensions $K \times 1$, and $3 \times K$ (three state) or $2 \times K$ (two state), respectively.

We will present one of many possible ways to study these GCNs corresponding to CA lookup tables by analyzing their largest singular value and associated normalized right singular vector of matrix $\mathbf{H}_1$, or the *leading eigenvector* associated of $\mathbf{H}_1^T\mathbf{H}_1$. In addition to the amount of data that is effectively produced from $\mathbf{H}_1$ being a large square matrix in our experiments, we also believe that since it is a feature transformation matrix between hidden layers, studying the spectral radius and associated eigenvector can inform what behaviors the GCN is modeling with respect to a given CA (e.g., large spectral radius perhaps means there is strong scaling of features along a low-dimensional manifold). The goal of this analysis being that we can produce conclusions amenable to a sort of principal component analysis (PCA) for weight matrices of these GCNs.

3.2.1. Bootstrap Architecture Variation. In the construction and training of GCNs there is a lot of variability in the choice of hyperparameters. Although the implicit architecture of the GCN is consistent in this lattice domain setting, fixing the number of states and neighborhood type, there is still a concern for the variability in the number of neurons to choose. In particular, there is a concern for whether there is any on the dependence by the CA neural profile on the parameter complexity of the corresponding GCN architecture. Addressing hypothesis **H1** we find that by Kolmogorov-Smirnov (KS) two-sample tests for differences in empirical cumulative distribution functions (CDFs) of the leading eigenvector of $\mathbf{H}_1^T\mathbf{H}_1$, that these distributions over 50 samples for varying $\mathbf{H}_1$ dimensions of $\left(1024 + 16 \times 2^{\{2,3,4,5,6\}}\right)^2$ are all significantly different for $p = 10^{-8}$ [**35**] [**36**] [**37**].

Figure 3.7, for BSP ($\tau = 0$) and BSP ($\tau = 7$) shows that the leading eigenvector CDFs have roughly the same shape, but due to the very large sample size for these eigenvector coordinates—smallest sample size is 1088×50 eigenvector entry values—the corresponding KS statistic produces an empirical value that is essentially machine epsilon. In other words, between these two CA, there is exists some parameter dependence on the eigenvectors of $\mathbf{W}^{(1)}$ for different CA. Hypothesis **H3** appears to hold for this neural profile characterization with eigenvectors. With this conclusion in mind, for the rest of this study we actually fix the number of hidden neurons for all CA in this four layer GCN at 2,048 since it is sufficiently large to easily fit different CA models, and again, since we should be able to extract rich data from a sufficiently large $\mathbf{H}_1$ hidden matrix.

3.2.2. Spectral Differences Between Cellular Automata Rules. Applying the same analysis as before, we find that for KS two-sample test, between all two-state CA on Moore neighborhoods and all three-state CA on von Neumann neighborhoods we can reject the hypothesis **H2**. More precisely, the leading eigenvector distributions $\mathbf{H}_1^T\mathbf{H}_1$ produces distinct distributions for our fixed two hidden layer GCN in 50 samples in both cases of two- and three-state CA.

Figure 3.8 for $\mathbf{H}_1 \in \mathbb{R}^{2048\times2048}$ shows that each leading eigenvector on average has a distinct distribution. Furthermore, we see more clearly that with respect to hypothesis **H2** the eigenvector distributions are all visually distinct as well (i.e., distinct CA rules appear to have distinct neural profile characteristics and we reject **H2**). Interestingly, we see that GOL is distinguished by a concentration of eigenvector values around $x = 0$. One possible reason for this being the case is that we can see that GOL singular values do not experience a sharp concave drop; rather, the second largest eigenvalue is similar in size to the spectral radius of $\mathbf{H}_1^T\mathbf{H}_1$ such that it is likely that unlike BSP, GOL transforms significant information along a two-dimensional manifold. A possible explanation of this behavior is that BSP strictly aggregates and sums surrounding nodes such that it only verifies whether a single threshold, τ, is met: *sum of 1-state neighbors is greater than* τ. GOL on the contrary checks that there are a certain number of 1-state neighbors, such that GOL can be characterized as having two thresholds for state transition 0 to 1, where the number of living neighbors much be between 3 or 4: *sum of 1-state neighbors is less than 5 and greater than 2.* The key point behind this observation is that the GCN could possibly be splitting the feature space of a two-state CA with respect to some thresholding behavior over neighborhood-state averages.

Figure 3.9 for three-state CA shows similar analysis and produces the same conclusion with respect to hypothesis **H2**—we reject **H2**. Each CCA and GHM is shown to have distinct eigenvector distributions, and the singular values experience a sharp *elbow* drop such that we can surmise that most of the information and variation of the weight matrix $\mathbf{H}_1$ is concentrated along a low-dimensional manifold. In the same figure we also include a reference "degenerate" CA corresponding to a rule where each neighborhood configuration maps to all three possible states: 0, 1 and 2. This degenerate CA corresponds to a random CA where we can judge GHM and CCA neural

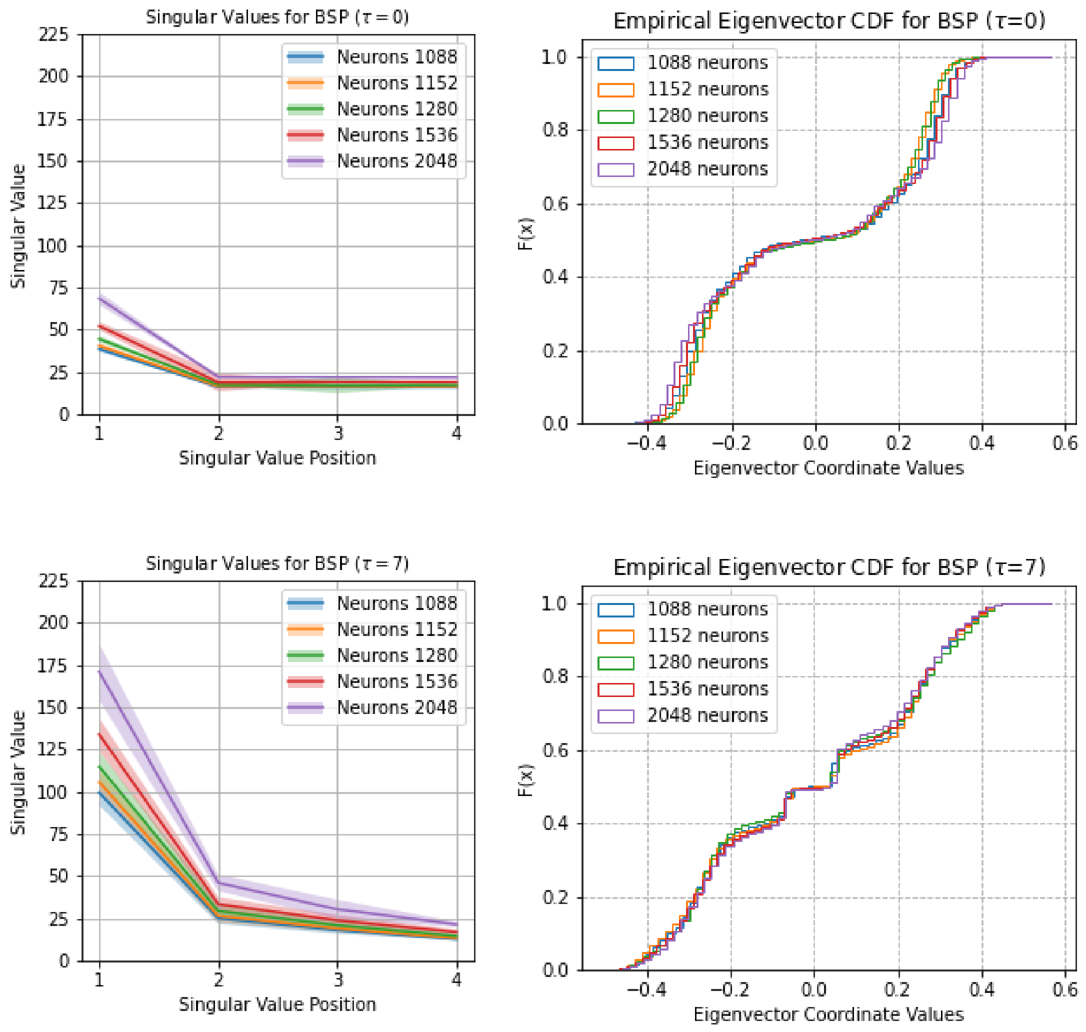

FIGURE 3.7. Left-to-right: average singular value by position and empirical CDF of leading eigenvector. Singular value distributions by position along x-axis and hidden layer parameter complexity by curve for $\tau = 0$ an $\tau = 7$. Each curve corresponds to the sample average singular value for 50 samples at each position with 95% confidence bands. The scale for the singular values for $\tau = 0$ appears to be very small, again, since it corresponds to the rule where no matter what each 0-state node becomes a 1-state node. Also, the corresponding empirical CDF plot appears to be bimodal as we can see a flat region around $x = 0$ where the CDF does not increase. There is significantly more variation in singular values for $\tau = 7$ by the singular value distribution plots. Similar to $\tau = 0$ there is a flat region around $x = 0$, but there is some mass distributed around $x = 0$ as well as a small jump in CDF.

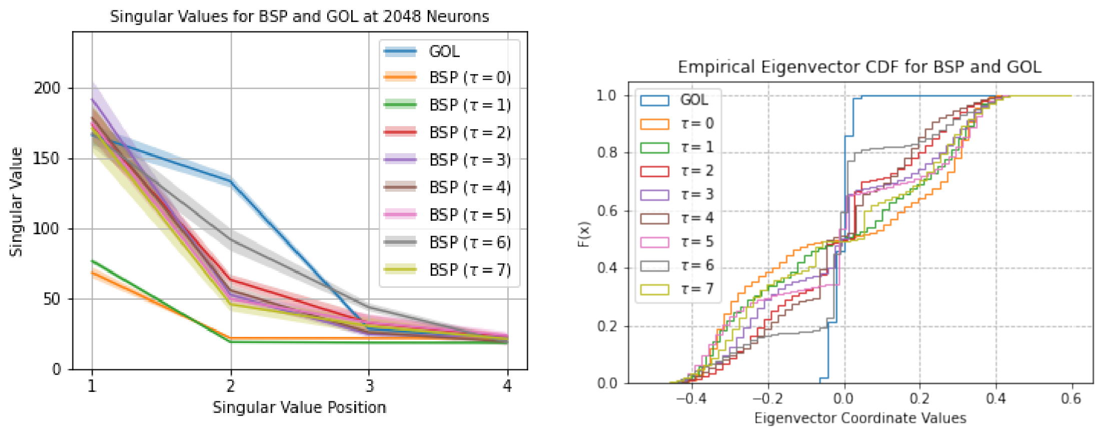

FIGURE 3.8. Left-to-right: Singular value distributions by position along x-axis and leading eigenvector CDF. Plots correspond to $\mathbf{H}_1 \in \mathbb{R}^{2048\times 2048}$ for BSP in all τ and GOL. Data for the average singular value distribution is computed with 50 samples of GCNs for each CA.

profiles against a degenerate neural profile (i.e., the hidden weight matrix singular values and leading eigenvectors consistently converge to a nontrivial distribution). Naturally, we see that this degenerate CA has random singular value distribution by their uniform appearance along the x-axis. In this case, with a KS two-sample test between GHM, CCA and the degenerate CA we find that the leading eigenvector distributions are all distinct. GHM is significantly different from CCA and the degenerate CA, and CCA is significantly different from the degenerate CA.

3.2.3. Damage Spread Stability and Spectral Analysis of Neural Profiles. There is currently no evidence for an association between damage spread—as a surrogate for stability—and neural profiles characterized by the largest singular value and leading eigenvector. A simple explanation for this is that the weight matrices of these neural networks only correspond to feature transformation properties of GCNs from a given neighborhood configuration into states at the next iteration, and not necessarily some global stability phenomenon. The singular value and singular vectors derived from these weight matrices only characterize properties of these weight matrices, and are only tangentially related, if at all, to emergent behaviors such as damage spread.

However, we note some observations between damage spreading and empirical spectral analysis of $\mathbf{H}_1$:

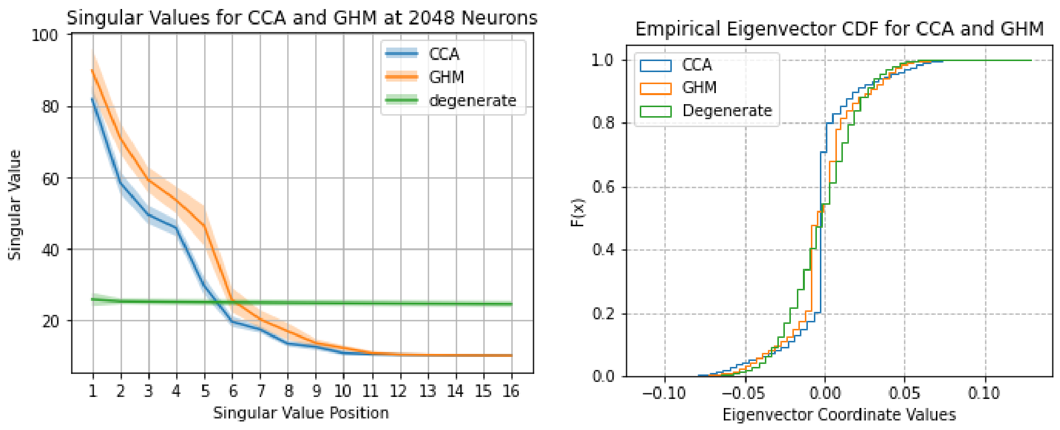

FIGURE 3.9. Left-to-right: Singular value distributions by position along x-axis and leading eigenvector CDF. Plots correspond to $\mathbf{H}_1 \in \mathbb{R}^{2048\times 2048}$ for CCA and GHM. Data for the average singular value distribution is computed with 50 samples of GCNs. In this case we also include a degenerate CA lookup table corresponding to a random rule such that this random rule can does not have a unique mapping for a single input (i.e., a lookup table where each input maps to three outputs randomly).

(1) For BSP there is no relationship between neural network characteristics and decreasing damage spreading stability characteristics through increasing threshold, τ. While the singular value distribution is smaller for $\tau = 0, 1$ in correspondence with strong damage spreading stability as seen in Figure 3.2, for $\tau = 2$, which also displays strong damage spreading stability, the singular value distribution is more dispersed and displaced along greater values.
(2) BSP ($\tau = 5$) and BSP ($\tau = 7$) have strong *elbows*—steep drop in singular values by singular value position—, while BSP ($\tau = 6$) has a more spread in singular values by position despite all three of these CA having weak stability.
(3) For CCA and GHM damage spreading is nearly the same as we see in Figure 3.6 and the corresponding singular value positions for these graphs are also similar as seen in Figure 3.9—in fact the average singular value curves are similar in shape as well.
(4) GOL is the only CA that exhibits damage spread that diffuses throughout the entire lattice and the only two-state CA such that it has two large, leading singular values. Again, this, in contrast to BSP for all τ where there is a strong elbow for the average singular value and damage spread is monotonically stable in time.

The aforementioned CA and their lookup tables are all learnable in 4 layers and 2,048 neurons with GCNs. However, four layer GCN model failed to learn the FCA lookup table on von Neumann neighborhoods with 2,048 neurons (certainly, it failed to learn with fewer neurons in each layer). Instead, it easily learned FCA when adding an additional layer, or 5 layer GCN, with an additional hidden matrix, $\mathbf{H}_2 \in \mathbb{R}^{2{,}048\times 2{,}048}$. With this apparent requirement for additional GCN complexity, we note that the average damage spread characteristics for FCA is considerably distinct since it exhibited oscillatory damage spread where CCA and GHM did not. Figure 3.10 shows the singular value distributions and leading eigenvector coordinate distributions of the two hidden matrices, $\mathbf{H}_1$ and $\mathbf{H}_2$. We again see that much more variation is contained in the first hidden matrix, and the second hidden matrix has fairly uniform singular value distributions. A soft explanation for this is that much of the feature transformation is handled between the first two hidden layers via the first hidden matrix, and the third hidden layer, or second hidden matrix, handles any residual feature information not handled by the first hidden matrix.

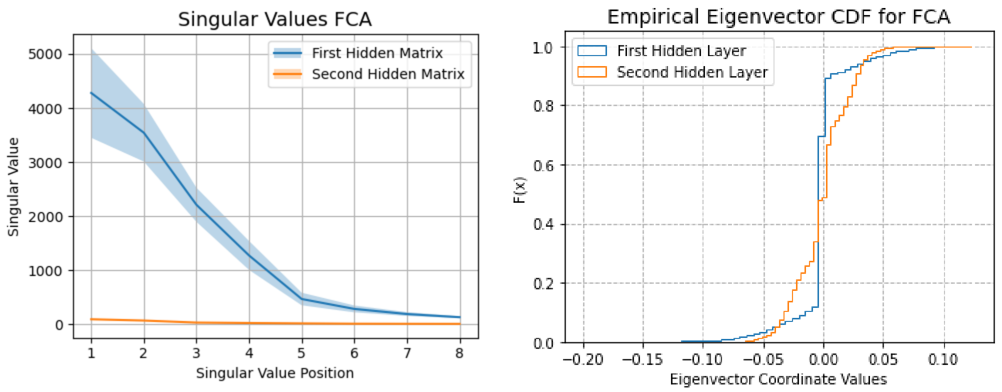

FIGURE 3.10. Left-to-right: Plot of singular value distribution of two hidden matrices and plot of corresponding leading eigenvectors coordinate distributions of each hidden matrix. Note that much of the variation is contained in the first hidden matrix indicating that the matrix is likely handling much of the feature transformation compared to the following hidden matrix.

With respect to hypothesis **H3** we reject the nonexistence of any relationship between CA stability and neural network profiles. In our case, we observe that damage spreading is one of many possible characteristics of CA stability and analysis of the largest singular values and leading eigenvectors of the first hidden matrix proves to be one of many possible ways to characterize neural profiles. For example, for GOL, CCA, GHM and FCA there appears to be some weak connection with damage spreading and leading eigenvector distributions. The main issue resides in BSP, where we observe a puzzling lack of relationship between damage spreading qualities and $\mathbf{H}_1$ neural profile characteristics as we increase τ. Despite the rejection of hypothesis **H3**, there are possibly different stability properties that can be better associated with alternative CA neural profiles characterizations.

CHAPTER 4

Conclusion

4.1. Summary of Results

In this thesis we have a developed a novel and flexible application of GCNs to study CA rules themselves under a single computational framework. Before this work, CA rules have been primarily studied through emergent behavior properties exhibited on various domains (e.g., random graphs, lattices, etc.).

With respect to the hypotheses from our introduction we have reached the following conclusions:

(**H1**) **Do not reject**: Neural profiles of CA are dependent on the parameter complexity of GCNs.

(**H2**) **Reject**: Distinct CA rules do not have different neural profile characteristics.

(**H3**) **Reject**: There is no association between CA stability and neural profile characteristics.

Our original purpose for posing **H1** was to hopefully argue that regardless of the hyperparameter complexity of a GCN for producing neural profiles, they would all ultimately have the same core characteristics. This claim seems to be apparent as evidence by the the similar shapes of empirical CDF curves of leading eigenvector coordinates of the BSP neural profiles. Yet, statistically these are all formally distinct CDFs. Even with this dependence of neural profiles to GCNs parameter complexity, this is still an acceptable situation as this neural profile tool can still be used to study and compare various CA fixing certain parameter choices.

Addressing **H2**, we found from KS two-sample tests on leading eigenvectors that all of these CA, both two-state and three-state CA, have distinct neural profile characteristics. This is a significant finding since it indicates that these neural profiles uniquely characterize the corresponding lookup tables modeled by our GCNs. Furthermore, despite showing unique neural profile characteristics, in cases where singular value distributions appear similar as we observed for CCA and GHM, we also observed similar average damage spreading characteristics in time.

Finally, we find that we can reject **H3** since there seems to be some evidence relating CA stability and neural profile characteristics. The most obvious example of this is the unique oscillatory average damage spreading of FCA and the learning difficulties of four layer GCNs for FCA. For CCA and GHM, they appear to show very similar average spreading dynamics, and even the corresponding average singular value curves appeared to have similar shapes. For BSP in varying τ and GOL, the leading eigenvector CDFs and damage spreading properties also appeared to be generally distinct. Again, our decision to reject **H3** is due to the fact there is some moderate evidence relating CA stability and neural profiles. However, there are other definitions of stability and neural profile characterizations to be explored.

4.2. Remarks

4.2.1. Limitations. With respect to the complexity of our data we note that we intentionally limit our study to at most three-state CA. Adding CA going beyond $\kappa = 3$ naturally increases the amount of data required to learn CA with greater κ since there is a requirement for the corresponding data to be upsampled for label balance. Again, the requirement for this upsample is because we require our GCN to learn each possible neighborhood configuration to label mapping with equal frequency (i.e., we must avoid bias towards any particular label). Similarly, we limit our study strictly to lattice domains due to this increasing complexity issue. Considering arbitrary domains there is a natural issue of neighborhood configurations of varying number of node neighborhood sizes (e.g., 1-node neighborhoods to n-node neighborhoods where $n \gg 1$).

Damage spreading, as flexible as it may be, is difficult to characterize and study analytically. Although a challenging task, it may be useful to develop an alternative stability characterization that also naturally incorporates local behaviors of CA. It may be best to simply develop a local stability characterization in which there would likely be much stronger evidence to assess validity **H1**. Perhaps a stability characteristic that measures the average rate of damage in time, or a measure of stability that measures defect accumulation could be more appropriate as well.

With respect to studying neural profiles, we could consider other numerical linear algebra techniques beyond what our PCA-style analysis. Other tools such as nonnegative matrix factorization

(NMF) could be useful to study these neural network layers instead [**38**]. We could also investigate the matrices adjacent to the input layer and output layer as there could be some interesting relationships embedded in these matrices corresponding to behavior of these CA rules as modeled by GCNs. The possibilities for modeling and analyzing discrete CA rules are endless and the main exciting point is that this process of utilizing GCNs can provide many new characterizations that are intrinsic to CA rules themselves. Thus we have developed an empirical tool to further general theory of cellular automata.

4.2.2. Extensions. Though we present an application of GCNs to CA, GCNs as utilized to capture local neighborhood mappings to state transitions has applications to general continuous dynamical systems. For example, we can learn a local neighborhood dynamical systems function of some real-world phenomenon by embedding data into a lattice-like, Euclidean domain. A dataset could be easily constructed as done above for CA, but much more elegantly since a GCN model would map continuous scalar values to continuous scalar values as opposed to a vector corresponding to discrete states. A lot of potential for utilizing GCNs in this way can potentially provide fruitful results in network science and dynamical systems literature.

APPENDIX A

Additional Figures

A.1. Average Damage Spreading Dynamics

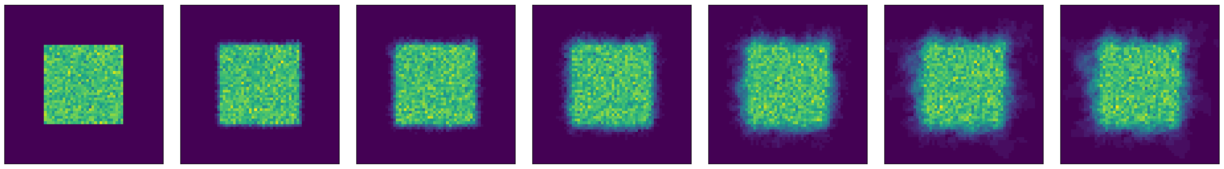

Figure A.1. From left-to-right: average damage spread dynamics of CCA for frame $t = 0, 2, 4, 8, 16, 32, 64$. For 30 samples of CCA the damage spread seems to appears to stabilize and stay confined within a small taxicab distance of 12 from the origin.

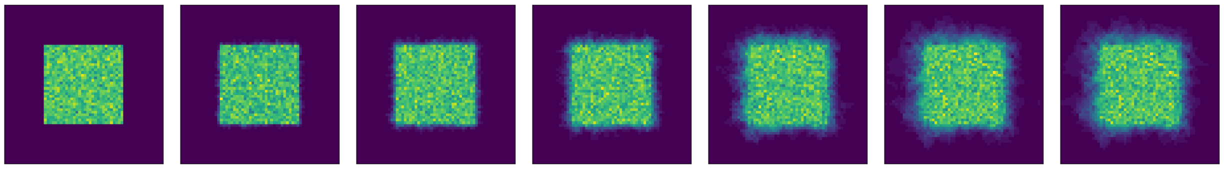

Figure A.2. From left-to-right: average damage spread dynamics of GHM for frame $t = 0, 2, 4, 8, 16, 32, 64$. For 30 samples of GHM the damage spread seems to appears to stabilize and stay confined within a small taxicab distance of 12 from the origin—we see similar behavior to CCA.

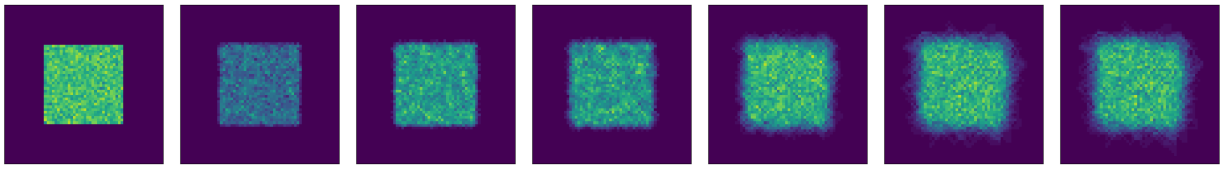

Figure A.3. From left-to-right: average damage spread dynamics of FCA for frame $t = 0, 2, 4, 8, 16, 32, 64$. FCA has a key characteristic of states "pulsing" as each vertex in the lattice oscillates through its phases and is inhibited by neighbors in the blinking color—hencing the damage appearing to dim as seen in frame $t = 2$. Otherwise, like CCA and GHM the damage ceases to spread for FCA. Interestingly, unlike CCA and GHM, the average damage spread dynamics for FCA will enter a sort of periodic orbit for all time such that the last frame is not static like CCA and GHM.

A.2. Empirical Spectral Analysis

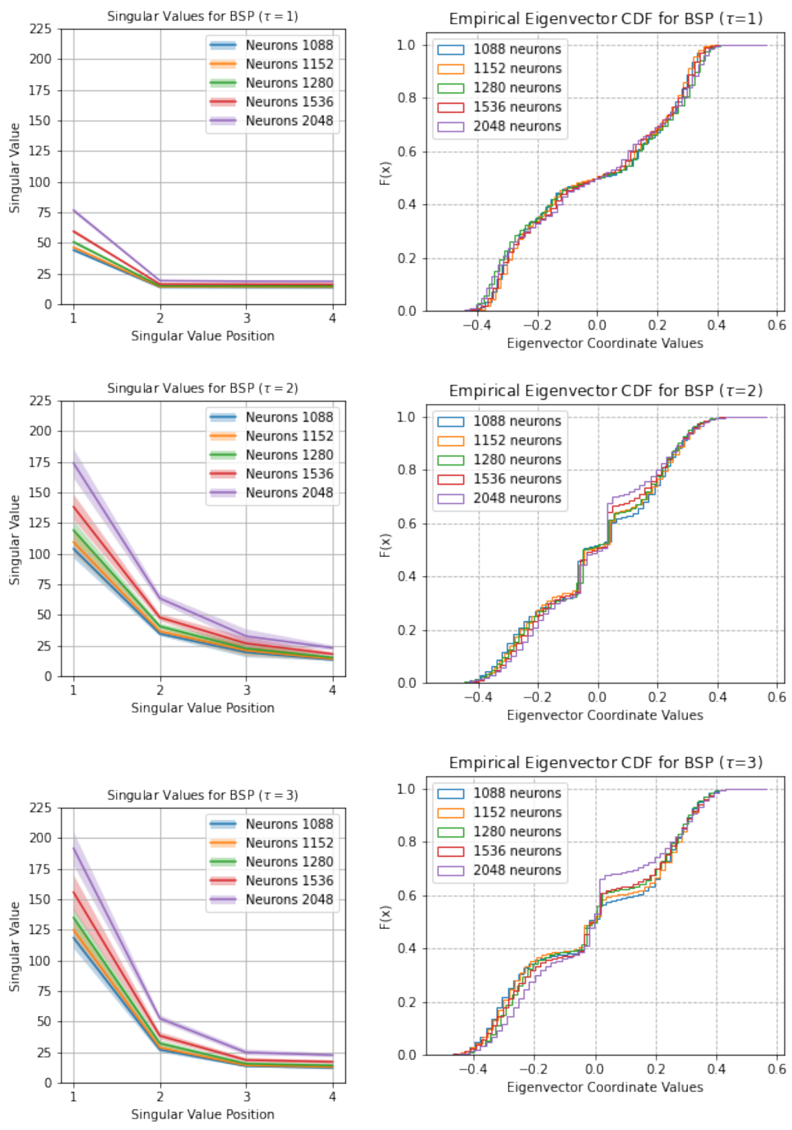

FIGURE A.4. Top-to-bottom: τ varying from 1 to 3. Left-to-right: Distributions of top three singular values and empirical CDF of eigenvector corresponding to spectral radius of $\mathbf{H}_1^T\mathbf{H}_1$ from first and only hidden layer. Note that figures for the singular value ranges are set at most 40,000. The scales for the singular values in $\tau = 1$ are much smaller than the singular values for $\tau = 2$. Despite Kolomogorov-Smirnov tests indicating distinct distributions, for 60 bins the empirical CDF by each τ has more or a less a consistent shape for varying hidden layer parameter complexity. (τ 3 to 6 is continued on the next page.)

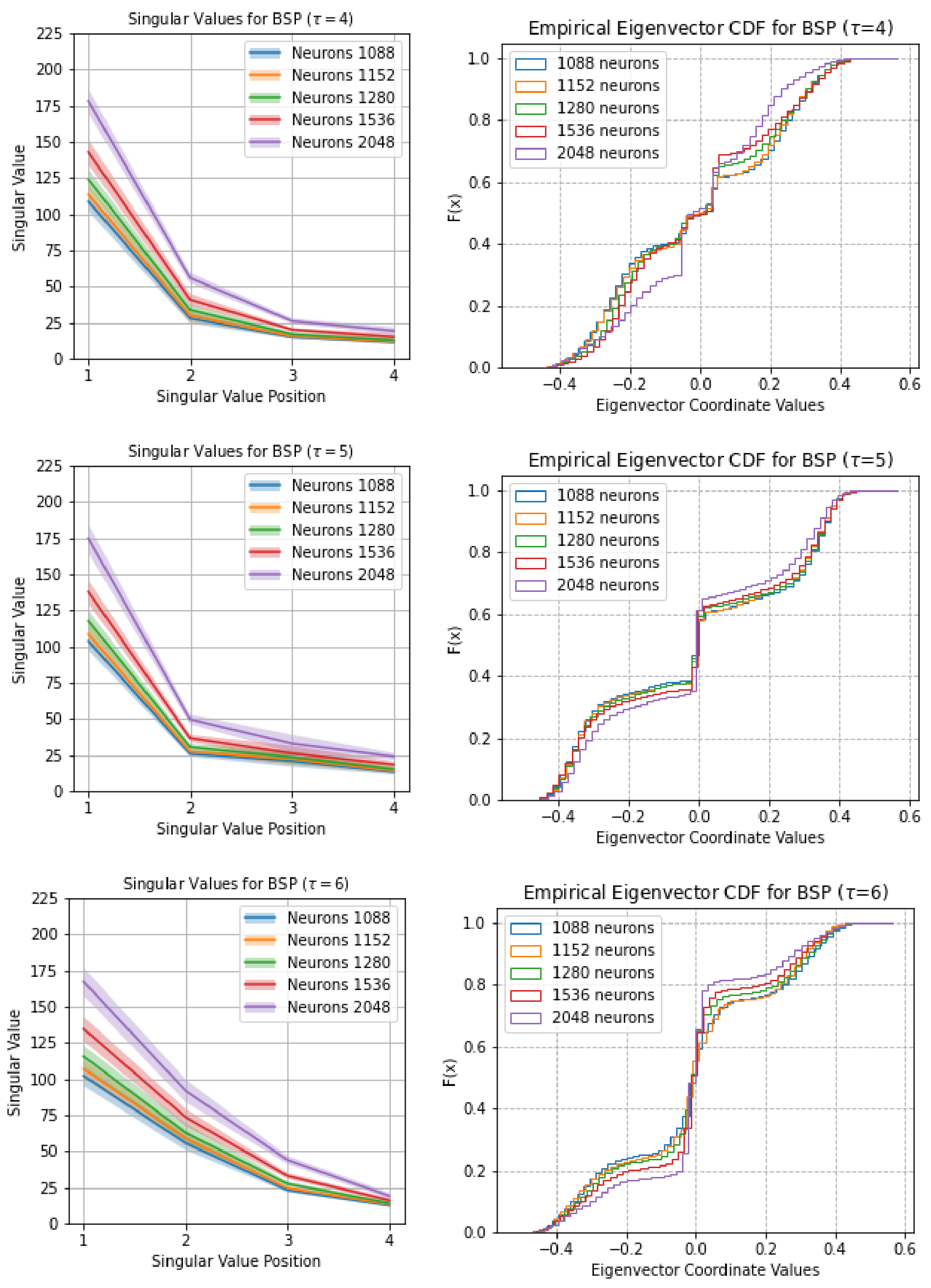

FIGURE A.5. (Continued figure) From top-to-bottom: varying $\tau = 4, 5$ and 6. The variation appears to be much more dispersed between the first two singular values unlike $\tau = 4$ and 5 as indicated by the roughly linear scales of singular values. Here we see that there is no proportional scaling in singular value distribution as we increase τ since we know that at $\tau = 7$ the sharp singular value *elbow* reappears.

Bibliography

[1] C. A. Pickover, *The math book: from Pythagoras to the 57th dimension, 250 milestones in the history of mathematics*. Sterling Publishing Company, Inc., 2009.

[2] S. Wolfram, "Statistical mechanics of cellular automata," *Reviews of modern physics*, vol. 55, no. 3, p. 601, 1983.

[3] J. M. Baetens and J. Gravner, "Introducing lyapunov profiles of cellular automata," *arXiv preprint arXiv:1509.06639*, 2015.

[4] J. M. Baetens and J. Gravner, "Stability of cellular automata trajectories revisited: branching walks and lyapunov profiles," *Journal of nonlinear science*, vol. 26, pp. 1329–1367, 2016.

[5] F. Dörfler, M. Chertkov, and F. Bullo, "Synchronization in complex oscillator networks and smart grids," *Proceedings of the National Academy of Sciences*, vol. 110, no. 6, pp. 2005–2010, 2013.

[6] S. K. Joshi, S. Sen, and I. N. Kar, "Synchronization of coupled oscillator dynamics," *IFAC-PapersOnLine*, vol. 49, no. 1, pp. 320–325, 2016.

[7] U. Biccari and E. Zuazua, "A stochastic approach to the synchronization of coupled oscillators," *Frontiers in Energy Research*, vol. 8, p. 115, 2020.

[8] S. Wolfram *et al.*, *A new kind of science*, vol. 5. Wolfram media Champaign, IL, 2002.

[9] J. Gravner, "Growth phenomena in cellular automata," *New constructions in cellular automata*, pp. 161–181, 2003.

[10] E. W. Weisstein, "Cellular automaton," *https://mathworld. wolfram. com/*, 2002.

[11] J. Adler, "Bootstrap percolation," *Physica A: Statistical Mechanics and its Applications*, vol. 171, no. 3, pp. 453–470, 1991.

[12] A. C. van Enter, "Proof of straley's argument for bootstrap percolation," *Journal of Statistical Physics*, vol. 48, pp. 943–945, 1987.

[13] E. R. Berlekamp, J. H. Conway, and R. K. Guy, *Winning ways for your mathematical plays, volume 4*. AK Peters/CRC Press, 2004.

[14] M. Gardner, "Mathematical games-the fantastic combinations of john conway's new solitaire game, life, 1970," *Scientific American, October*, pp. 120–123, 1970.

[15] R. Fisch, "The one-dimensional cyclic cellular automaton: a system with deterministic dynamics that emulates an interacting particle system with stochastic dynamics," *Journal of Theoretical Probability*, vol. 3, pp. 311–338, 1990.

[16] R. Fisch, J. Gravner, and D. Griffeath, "Threshold-range scaling of excitable cellular automata," *Statistics and Computing*, vol. 1, pp. 23–39, 1991.

[17] J. M. Greenberg and S. P. Hastings, "Spatial patterns for discrete models of diffusion in excitable media," *SIAM Journal on Applied Mathematics*, vol. 34, no. 3, pp. 515–523, 1978.

[18] R. Durrett and J. E. Steif, "Some rigorous results for the greenberg-hastings model," *Journal of Theoretical Probability*, vol. 4, pp. 669–690, 1991.

[19] R. Fisch, J. Gravner, and D. Griffeath, "Metastability in the greenberg-hastings model," *The Annals of Applied Probability*, vol. 3, no. 4, pp. 935–967, 1993.

[20] J. Gravner, H. Lyu, and D. Sivakoff, "Limiting behavior of 3-color excitable media on arbitrary graphs," *The Annals of Applied Probability*, 2018.

[21] H. Lyu, "Synchronization of finite-state pulse-coupled oscillators," *Physica D: Nonlinear Phenomena*, vol. 303, pp. 28–38, 2015.

[22] H. Bassi, R. P. Yim, J. Vendrow, R. Koduluka, C. Zhu, and H. Lyu, "Learning to predict synchronization of coupled oscillators on randomly generated graphs," *Scientific Reports*, vol. 12, no. 1, p. 15056, 2022.

[23] J. M. Springer and G. T. Kenyon, "It's hard for neural networks to learn the game of life," in *2021 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8, IEEE, 2021.

[24] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[25] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT press, 2016.

[26] W. Zhang, J. Tanida, K. Itoh, and Y. Ichioka, "Shift-invariant pattern recognition neural network and its optical architecture," in *Proceedings of annual conference of the Japan Society of Applied Physics*, vol. 564, Montreal, CA, 1988.

[27] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[28] B. Sanchez-Lengeling, E. Reif, A. Pearce, and A. B. Wiltschko, "A gentle introduction to graph neural networks," *Distill*, vol. 6, no. 9, p. e33, 2021.

[29] J. Zhou, G. Cui, S. Hu, Z. Zhang, C. Yang, Z. Liu, L. Wang, C. Li, and M. Sun, "Graph neural networks: A review of methods and applications," *AI open*, vol. 1, pp. 57–81, 2020.

[30] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[31] A. F. Agarap, "Deep learning using rectified linear units (relu)," *arXiv preprint arXiv:1803.08375*, 2018.

[32] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," in *Proceedings of the IEEE international conference on computer vision*, pp. 1026–1034, 2015.

[33] M. Fey and J. E. Lenssen, "Pytorch geometric." `https://github.com/rusty1s/pytorch_geometric`, 2019.

[34] K. García-Medina, D. Estevez-Moya, and E. Estévez-Rams, "Damage spreading and information distance in cellular automata," *Revista Cubana de Física*, vol. 39, no. 2, pp. 90–97, 2022.

[35] V. W. Berger and Y. Zhou, "Kolmogorov–smirnov test: Overview," *Wiley statsref: Statistics reference online*, 2014.

[36] K. AN, "Sulla determinazione empirica di una legge didistribuzione," *Giorn Dell'inst Ital Degli Att*, vol. 4, pp. 89–91, 1933.

[37] N. V. Smirnov, "On the estimation of the discrepancy between empirical curves of distribution for two independent samples," *Bull. Math. Univ. Moscou*, vol. 2, no. 2, pp. 3–14, 1939.

[38] D. D. Lee and H. S. Seung, "Learning the parts of objects by non-negative matrix factorization," *Nature*, vol. 401, no. 6755, pp. 788–791, 1999.