Unsupervised Learning in Complex Systems

Hugo Cisneros

PSL
UNIVERSITÉ PARIS

THÈSE DE DOCTORAT
DE L'UNIVERSITÉ PSL

Préparée à L'École normale supérieure de Paris

Unsupervised Learning in Complex Systems

Soutenue par

Hugo Cisneros

Le 15 Mai 2023

École doctorale n°386

Sciences Mathématiques de Paris Centre

Spécialité

Informatique

Composition du jury :

Lenka Zdeborová École Polytechnique Fédérale de Lausanne	*Président du jury*
Stefano Nichele Østfold University College	*Rapporteur*
Hiroki Sayama Binghamton University	*Rapporteur*
Josef Sivic École Normale Supérieure, Inria CIIRC, Czech Technical University	*Directeur de thèse*
Tomas Mikolov CIIRC, Czech Technical University	*Co-encadrant de thèse*

1794
ENS | PSL

Résumé

Dans cette thèse, nous explorons l'utilisation de systèmes complexes pour étudier l'apprentissage et l'adaptation dans les systèmes naturels et artificiels. L'objectif est de développer des systèmes autonomes capables d'apprendre sans supervision, de se développer de manière autonome et de devenir de plus en plus complexes avec le temps. Les systèmes complexes apparaissent comme un cadre adapté pour comprendre ces phénomènes en raison de leur capacité à croître en complexité. Être en mesure de construire des algorithmes d'apprentissage qui nécessitent peu ou pas de supervision permettrait une plus grande flexibilité et adaptabilité dans diverses applications. En tentant de comprendre les principes fondamentaux de l'apprentissage dans les systèmes complexes, nous espérons améliorer notre capacité à concevoir et à mettre en œuvre des algorithmes d'apprentissage à l'avenir. Cette thèse apporte les contributions suivantes : le développement d'une métrique de complexité générale que nous appliquons pour rechercher des systèmes complexes qui présentent une croissance de complexité, l'introduction d'une méthode pour étudier les calculs dans des systèmes complexes à grande échelle, et le développement d'une métrique pour l'efficacité de l'apprentissage ainsi que l'introduction d'un jeu de données de référence pour évaluer la vitesse des algorithmes d'apprentissage. Nos résultats contribuent significativement à notre compréhension de l'apprentissage et de l'adaptation dans les systèmes naturels et artificiels. De plus, notre approche contribue à une nouvelle direction prometteuse pour la recherche dans ce domaine. Nous espérons que ces résultats inspireront le développement d'algorithmes d'apprentissage plus efficaces et plus performants à l'avenir.

Apprentissage automatique ⋆ Intelligence artificielle ⋆ Systèmes complexes ⋆ Complexité

Abstract

In this thesis, we explore the use of complex systems to study learning and adaptation in natural and artificial systems. The goal is to develop autonomous systems that can learn without supervision, develop on their own, and become increasingly complex over time. Complex systems are identified as a suitable framework for understanding these phenomena due to their ability to exhibit growth of complexity. Being able to build learning algorithms that require limited to no supervision would enable greater flexibility and adaptability in various applications. By understanding the fundamental principles of learning in complex systems, we hope to advance our ability to design and implement practical learning algorithms in the future. This thesis makes the following key contributions: the development of a general complexity metric that we apply to search for complex systems that exhibit growth of complexity, the introduction of a coarse-graining method to study computations in large-scale complex systems, and the development of a metric for learning efficiency as well as a benchmark dataset for evaluating the speed of learning algorithms. Our findings add substantially to our understanding of learning and adaptation in natural and artificial systems. Moreover, our approach contributes to a promising new direction for research in this area. We hope these findings will inspire the development of more effective and efficient learning algorithms in the future.

Machine learning ⋆ Artificial Intelligence ⋆ Complex systems ⋆ Complexity

Acknowledgments

I express my sincere gratitude to Professor Josef Sivic and Tomas Mikolov for their invaluable guidance, enthusiasm, and endless encouragement, without which this thesis would not have been possible.

I am very grateful to a number of people who have directly or indirectly influenced and helped me in my work, both as colleagues and friends: Jelle, Barbora, Kateryna, Teven, and David. Special thanks to Barbora Hudcová and Josef Sivic for their meticulous proofreading of parts of this thesis.

I would also like to thank everyone at the Foundational AI Lab and the Automated Reasoning Lab of the Czech Institute of Informatics, Robotics and Cybernetics (CIIRC) for creating a welcoming and supportive work environment.

Most importantly, I would like to express my deep appreciation to my parents, family, and friends for their support and understanding throughout my academic journey.

Lastly, I would like to express my heartfelt thanks to Brune for her unwavering patience, help, support, and constant inspiration.

Contents

Chapter

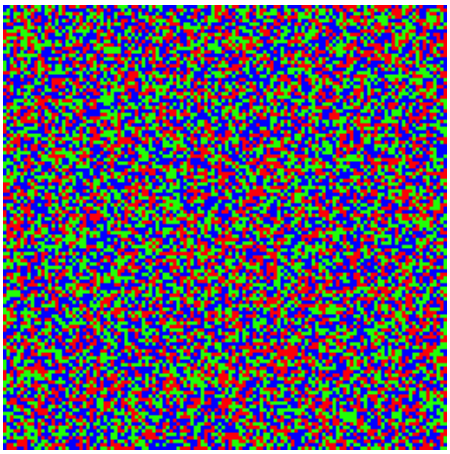
1 Introduction

1.1 Goal

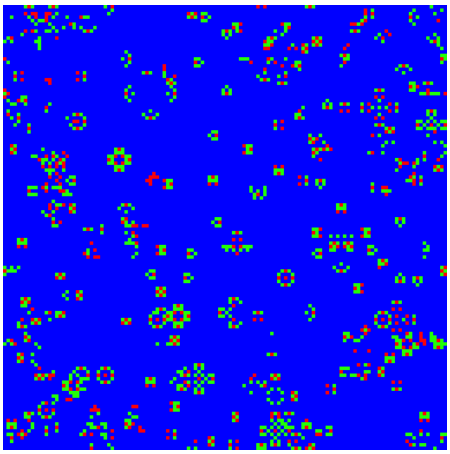
Complex systems are systems composed of many interconnected elements that interact with each other, often in non-linear ways, giving rise to emergent properties that cannot be explained by the properties of the individual elements alone. In many naturally occurring complex systems, learning is a key process that allows the system to adapt and evolve over time. In this thesis, we explore complex systems as a framework for studying learning and adaptation in natural and artificial systems. The aim of this thesis is to develop methods for studying and using computations that take place in complex dynamical systems to eventually create learning algorithms that require limited to no supervision. Complex systems often exhibit features such as self-organization, adaptation, feedback loops, and the ability to undergo phase transitions, all of which can make them challenging to model and predict. Examples of complex systems that occur naturally include ecosystems, weather patterns, financial markets, social networks, and biological organisms. The objective of this thesis is broken down into the following subgoals:

1. The first subgoal is to identify complex dynamical systems that have the potential to display emergent open-ended growth, which refers to the ability of a system to spontaneously generate new levels of complexity over time without reaching a state of equilibrium. This growth is often associated with evolutionary-like properties, such as variation, selection, and inheritance. There are many ways to define complex systems, and as illustrated in Figure 1.1, some may exhibit more interesting and promising behaviors than others. A cellular automaton (CA) is an instance of a complex system that can be described as a grid of cells that can take on a finite set of states and change their state over time according to

a set of rules that depend on the states of their neighboring cells. Interesting Cellular automata (CAs), such as the one shown in Figure 1.1b may be hard to find depending on how the search space is defined. The work presented in Chapter 4 aims to construct a metric of complexity that can help to identify interestingly behaving complex dynamical systems.



(a) A disordered cellular automaton.



(b) A cellular automaton with visible emergent structures.

Figure 1.1 – Two examples of complex systems (cellular automata) with different behavior types. 1.1b and 1.1a show a single state of a randomly initialized 2D CA simulated for a fixed number of steps. Some CAs appear more promising than others for the design of unsupervised learning systems because of their emergent complex structures (visible in 1.1b), whereas the CA in 1.1a seems to behave randomly.

2. The second subgoal is to measure the fraction of systems that have the most complex and rapidly evolving behavior. Defining these notions is also part of the goal. This sub-goal is distinct from the first one, which focuses on complex system design in general. Instead, this sub-goal involves establishing approaches to evaluate growth in complexity. We believe that systems with the most complex and rapidly evolving behavior are promising for further use since they may exhibit open-ended complexity growth. In Chapters 4 and 5, we present different methods to measure the evolving complexity and understand when the complexity increases over time. Chapter 5 of this thesis is dedicated to investigating the importance of multiscale analysis for the complexity of cellular automata, which is the process of studying a system at different levels of detail or resolution. Additionally, we identify complex systems with behavior that changes when we manipulate the

scale of the system, either by increasing or decreasing its size.

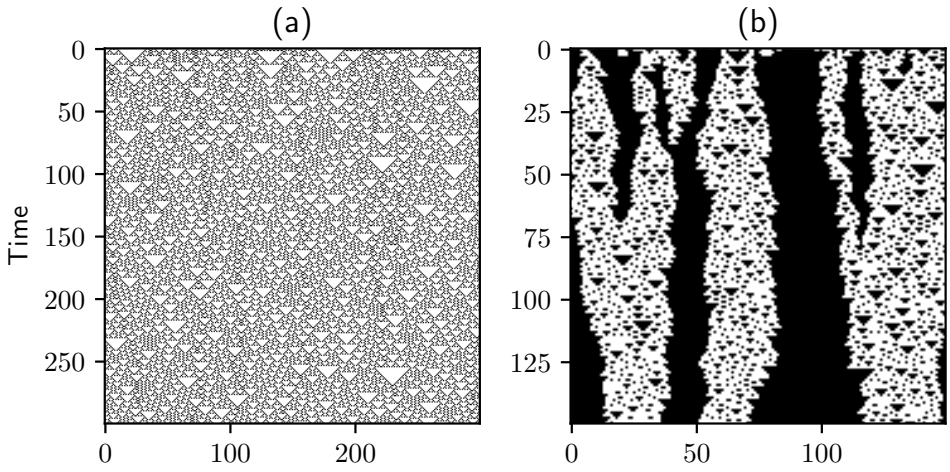


Figure 1.2 – Filtering the behavior of elementary cellular automaton rule 18. This uncovers a highly structured behavior with the propagation of area boundaries within the apparent randomness. (a) shows 300 timesteps of a randomly initialized rule 18 simulation. Notice the complex structures made visible in (b) with a filtering method. This Figure is discussed in more detail in Chapter 5.

3. Apply promising systems to challenging learning tasks where classical machine learning models may fail or become less efficient. The goal is to define various ways to apply evolving complex dynamical systems to some standard learning tasks and to find out if it improves the performance or efficiency of the learning algorithm. An example application that we explore in Chapter 6 is illustrated in Figure 1.3, where a CA is used to implement a language model, a probabilistic model that is used to generate text to complete sentences. In Chapter 2, we explore the similarities between CAs networks and highlight the numerous potential applications of CAs.

In this thesis, we do not focus our attention on what the machine learning community commonly refers to as "unsupervised learning", that is, learning underlying structures from unlabeled data (Hinton and Sejnowski 1999). In machine learning, supervision refers to a label, a number, or a collection of numbers that represents an expected outcome associated with some input data. Our goal is to construct models that can develop autonomously without necessarily needing any external data or interactions. Therefore, we first seek systems that behave that way on their own and postpone the issue of learning to optimize a particular objective function to the end of the thesis

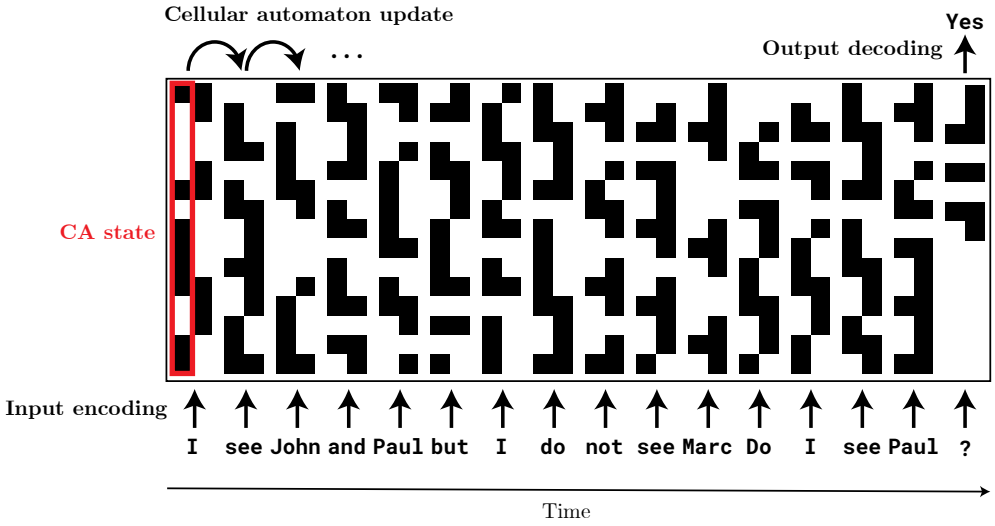


Figure 1.3 – An example of using a cellular automaton and reservoir computing to implement a language model. Tokens are encoded within the cellular automaton internal state. The cellular automaton update rule is then applied, and an output value is decoded from the final state. Each vertical bar represents two consecutive internal CA states, which are enriched with the encoded input.

(Chapter 6). We are particularly interested in systems that can develop through internal evolution rules without any input, which is the case for many complex systems.

Throughout the thesis, we work toward achieving these goals, focusing on one particular complex system: the cellular automaton (Von Neumann and Burks 1966). This model has been extensively studied due to its simple definition and its ability to simulate a wide range of complex behaviors. We provide a detailed description of cellular automata in Section 2.1. This thesis provides new insights into the role of learning in complex systems and open-ended evolution and demonstrates the potential of cellular automata as a model for studying these phenomena.

1.2 Motivation

It is possible that some form of evolutionary mechanism may be necessary in order to achieve advanced forms of artificial intelligence (AI), particularly if the goal is to create intelligent systems that can adapt and learn in complex and changing environments. Complex dynamical systems could be the key to overcoming the problems with existing learning algorithms, such as difficulties in generalization, robustness, or the ability to learn continuously (Parisi et al. 2019). The natural intelligence of biological systems

seems to depend on emerging properties selected through evolution. For example, biological life exhibits a pattern of major evolutionary transitions in which autonomously replicating entities at the lower level merged to form a single more complex corporate body (Lorenz, Jeng, and Deem 2011). This is believed to have occurred for the emergence of eukaryotic cells, as well as for multicellular life (Hammerschmidt et al. 2014). All complex and diverse biological entities have presumably emerged from a single common ancestor and, even before, from inorganic components present on the surface of the Earth (J. M. Smith and Szathmary 2000; Woese 1998).

So far, it remains uncertain which algorithmic characteristics might enable an artificial system to exhibit a path comparable to the natural evolutionary process within its state space, which involves the transition from basic constituents to intricate entities. Producing emerging phenomena similar to those of nature *in silico* is a long-standing challenge. Many algorithms try to mimic evolutionary properties to solve particular tasks, called evolutionary algorithms (Bäck and Schwefel 1993; Fogel, Owens, and Walsh 1966; Miller, Todd, and Hegde 1989). However, most of these methods focus on searching the space of solutions using high-level evolutionary mechanisms, such as genetic mutations and crossovers. Although effective in solving precise tasks, these methods often obscure another crucial component of natural evolution by using an explicit fitness function and hard-coded primitives.

Most existing machine learning algorithms rely on the choice of an objective function: a clearly defined mapping from the current state and parameters of a model to a real value, which indicates the performance of that model. The function depends on the objective of the model. For a supervised learning problem, we may count the number of misclassified objects or the distance between the predictions and the expected results. Even in unsupervised learning, the family of algorithms used for learning from unlabeled data, objectives are still central. For example, the well-known K-means clustering algorithm minimizes the sum of square distances of data points to cluster centers.

This reliance on objective functions creates two main issues: (i) The objective is not always clearly defined or can be too broad for general-purpose applications. For example, a possible objective function of a walking robot could be “not fall when stepping through its surrounding environment”. This function is impractical to define and will vary

greatly depending on the parameters of the environment (e.g. depending on the terrain, malfunctioning or missing limbs of the robots). Indirect rewards such as "head not touching the ground" may help, but it does not cover all possible ways the model could fail. This problem of defining an objective is also present within our goal of designing a general-purpose, autonomously developing learning algorithm. Furthermore, (ii) using predefined functions as goals can be counterproductive because, as many examples in nature demonstrate, robust paths to complex objectives are often deceptive. They involve developing in unexpected directions that may initially seem to be against the original goal (Stanley and Lehman 2015).

In this thesis, the term *unsupervised* refers to a form of learning with no predefined objective. Like in natural evolution, we expect true unsupervised algorithms to develop new features autonomously and become progressively more complex over time. Such algorithms would regularly learn to solve problems on their own without the need to be explicitly guided towards the problem solutions, thereby discovering robust and diverse solutions to deceptive problems.

1.3 Challenges

The study of complex systems presents a range of difficult challenges by itself (San Miguel et al. 2012). The complexity of a system is an emergent property that arises from various factors, including its intricate structure, the number of elements it contains, how it functions, and how it responds to different types of external influences. These factors contribute to the overall complexity of the system, which could be measured in various ways. For many complex systems, their emergent mechanisms are poorly understood. We identify four main challenges associated with the study of complex systems in the context of this thesis: (i) the questions of the design choices in defining and sampling a complex system (section 1.3.1), (ii) which will in turn define their potential to support a form of open-ended evolution (section 1.3.3) and (iii) how we can expect to build an interface to communicate with it (section 1.3.4), (iv) which is essential to achieve some form of control of that system (section 1.3.2).

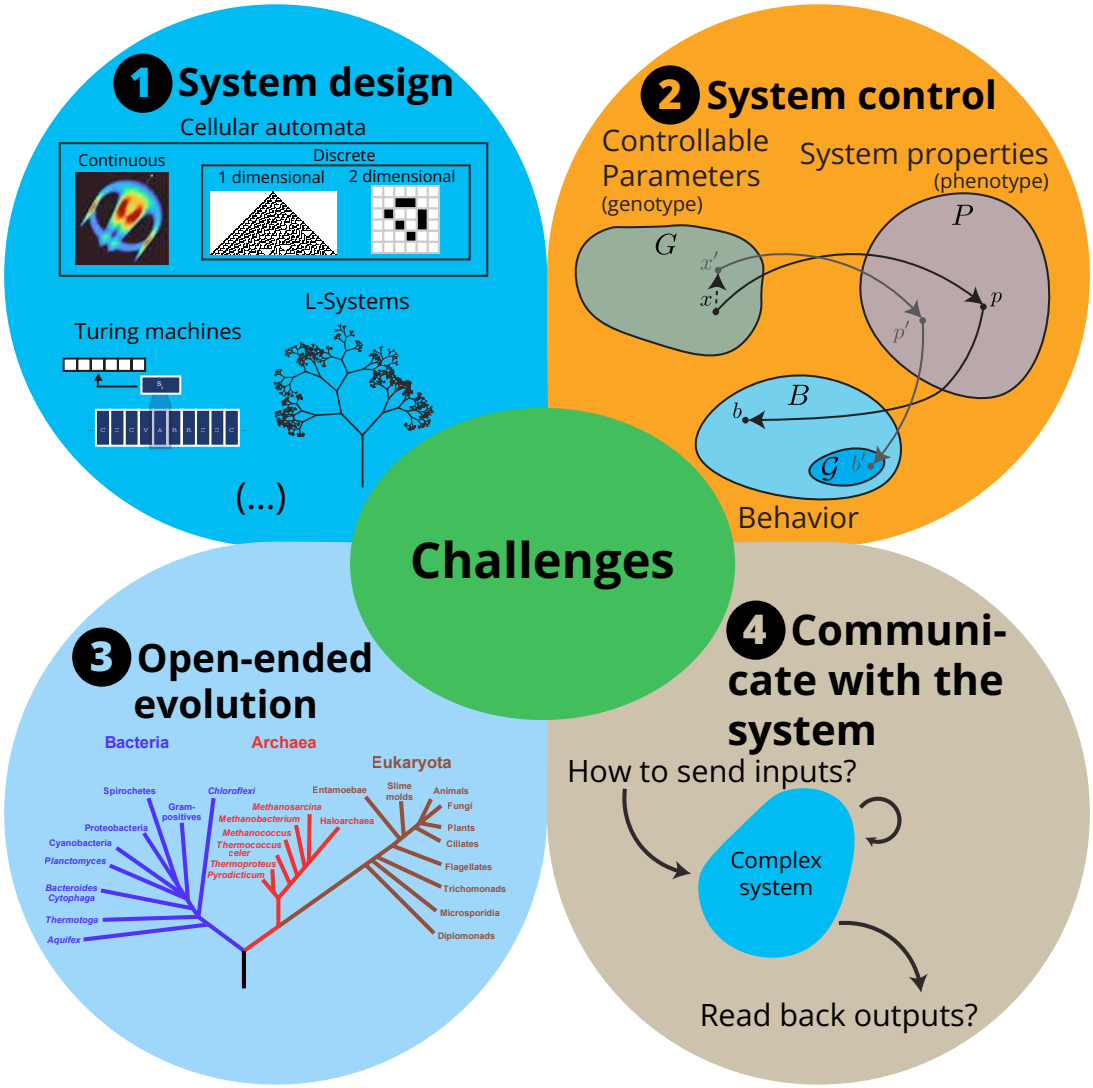


Figure 1.4 – The challenges of working with complex systems encountered across this thesis can be broken down into several categories. **(1)** The choice and design of the system, **(2)** the control of the system which involves understanding the complex mapping between a parameter space and the possibly unpredictable behavior of a system, **(3)** the search for open-ended evolution properties similar to the ones found in nature, **(4)** the issue of sending inputs and reading outputs from the internal state of a complex system. This illustration uses Lenia icon4.png by Bert Wang-Chak Chan, licensed under CC BY 4.0.

1.3.1 Design of a complex system

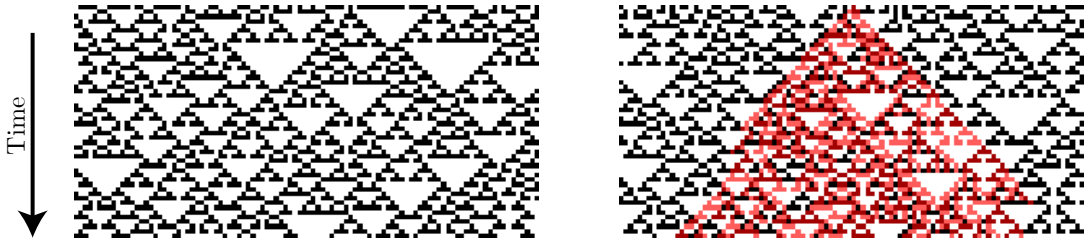
The first challenge is to construct a suitable complex system. The definition of complex systems is broad, and several systems with interesting dynamics have been studied under that name. For example, abstract models such as L-Systems (formal grammars used to model the growth and development of organic structures), Random Boolean networks (dynamical systems consisting of a fixed number of binary nodes that are randomly connected to each other and updated based on the state of its neighboring nodes), Turing machines (abstract deterministic computing machines consisting of a tape and a read-write head) and Cellular automata (CAs). In this thesis, we focus mainly on the last listed element: the cellular automaton. There are multiple benefits to working with this model. It is very simple to define, and its high parallelism makes its implementation straightforward. Furthermore, CAs have shown the ability to simulate a wide range of complex behaviors (Wolfram 2002).

Choosing the right complex system architecture is essential because it defines the search space over which interesting and useful systems can be found. Correctly parameterizing that space can also be challenging because too many degrees of freedom make it difficult to search for and find good systems, while too few might indicate a lack of expressivity. An example parameterization of CAs with little expressivity is Langton's lambda parameter (Langton 1990). At the other end of the spectrum, the CA rule is a simple parametrization with many degrees of freedom, which makes it impractical.

Even when we limit ourselves to the study of CAs, many variants can be considered (we list some of them in Chapter 2). They can have continuous or discrete states and operate in continuous or discrete space. For discrete state and space CAs, the number of states and the window of the update function can vary, as well as the topology of the simulation space. We tackle this issue from various angles throughout the thesis, exploring the question of rule definition in Chapter 4, the scale in Chapter 5, and the connectivity pattern in Chapter 6.

1.3.2 Complex systems control

Even if we want complex systems that evolve in unexpected directions and grow in an open-ended way without supervision, it may still be useful to steer them locally toward



(a) CA rule 22 on a tape of size 100, ran for 50 steps from a random initial state.

(b) CA rule 22 on a tape of size 100 ran from a slightly different initial state. The cells different from Figure 1.5a are overlayed in red.

Figure 1.5 – Comparison of the same 1 dimensional CA ran from two initial conditions differing only by one cell. Each row is the state of the CA at one time step, with time increasing from top to bottom. The effect of the small perturbation of initial conditions grows rapidly as the CA evolves in time.

specific targets. A major challenge that follows from this goal is that many complex systems are unpredictable. Deterministic dynamical systems can exhibit behavior that is extremely sensitive to their initial conditions. This sensitivity is sometimes referred to as chaos, where even tiny perturbations in the initial conditions can result in significantly different outcomes or trajectories. This is illustrated with a simple CA rule in Figure 1.5. Because of this property, it is often very hard to predict how a given system will evolve over time, and therefore hard to steer its evolution towards a particular final state or along a chosen trajectory. Although it can be challenging to apply these systems to certain basic tasks, their ability to generate unexpected solutions to difficult problems is also an advantage for the development of open-ended systems. Such systems need to continuously produce new and diverse solutions without a predefined endpoint and their sensitivity to initial conditions can be helpful in achieving this goal.

This problem is also related to the issue of sending a control signal to a complex system that depends on the choice of encoding for that signal. This encoding can significantly impact the system's ability to respond to the control signal and generate the desired outcome, which we describe in Section 1.3.4.

1.3.3 Open-ended evolution without objectives

Another challenge is posed by the lack of a clear objective in the design of fully unsupervised learning systems. Even without an objective, it is still essential to understand which complex systems to choose from all available options or how to tune their

parameters so that they behave in interesting ways. For example, CAs such as the one shown in Figure 1.1a is unlikely to be useful for building systems that demonstrate a growth of complexity due to their disordered nature in both space and time. Because of this disordered behavior, they cannot preserve information about the past.

Our goal is to create a system that has the property of evolving in an open-ended way, and we need metrics that can help select systems. However, these metrics should not be used as an additional objective function, as doing so would result in the problems described in Section 1.2. We address this challenge in particular in Chapter 4, by designing a complexity metric that can help select interesting complex systems without relying on a specific task-based performance score.

1.3.4 Complex systems inputs and outputs

Some of the complex systems we study in this thesis are closed systems. They do not expect inputs or have well-defined outputs. For example, CA and random boolean network (RBN) do not have the notion of inputs and outputs built into the model. They are standalone objects that evolve according to a set of internal rules.

A common challenge for such systems is to define the inputs and outputs in a way that preserves their internal dynamics. This is also related to the control problem of Section 1.3.2, because controlling a complex system implies being able to send a control signal and read the current state of the system.

Throughout this work, we use a framework called *reservoir computing* (RC) for this purpose. Reservoir computing allows one to harvest the internal computations of complex systems, or *read* information from its internal state by learning a linear regression that maps that internal state to desired outputs (see Section 2.3).

1.4 Contributions

The main contributions of this thesis are as follows.

1. Our first contribution is a review of the literature (Chapter 3) describing the connections between cellular automata and other complex systems, open-ended evolution, and neural networks. Studying these fields from a single point of view

is a relatively novel endeavor, and we consider a review necessary to place the rest of our work in its context.

2. Our second contribution is the development of a general complexity metric that can help identify complex systems with interesting behavior (Chapter 4). Our findings were published in (Cisneros, Sivic, and Mikolov 2019).

3. The third contribution is the development of a coarse-graining method to visualize computations in cellular automata and other discrete systems with local interactions (Chapter 5). Our findings were published in (Cisneros, Sivic, and Mikolov 2020).

4. Our fourth contribution is the introduction of a learning efficiency metric for learning algorithms and a benchmark dataset of progressively harder language tasks (Chapter 6). Our findings were published in (Cisneros, Mikolov, and Sivic 2022).

1.5 Thesis overview

Chapter 2 presents some background notions about complex systems, cellular automata, and reservoir computing.

In Chapter 3, we review relevant methods and tools for measuring the complexity of complex systems and use their computations for various tasks.

Chapter 4 introduces a complexity metric that allows us to select complex systems with interesting behavior. The metric measures the "novelty" of the temporal states of a system compared to a reference system. We built a dataset of interesting cellular automata to validate the quality of the metric.

Chapter 5 addresses the question of large-scale complex systems and the applicability of complexity metrics at multiple scales. We propose three algorithms for the coarse-graining of cellular automata. This allows us to reduce the size of large-scale systems while retaining the interesting parts of the behavior.

Chapter 6 presents a learning efficiency metric and a dataset to measure the learning speed of various systems. We show that reservoir computing-based systems using cellular automata can be more efficient than usual machine learning algorithms in constrained

data and computation settings.

Chapter 7 summarizes our contributions to this thesis.

Chapter 8 outlines potential avenues for further research in the field. It provides suggestions on how current work could be expanded or improved, highlighting areas that require additional research to advance understanding of the subject. We propose some experiments that could extend the potential applications of cellular automata, as well as a theoretical framework for learning in dynamical systems.

1.6 Publications and software

The thesis has led to the following publications.

— Hugo Cisneros, Josef Sivic, and Tomas Mikolov (Dec. 2019). « Evolving Structures in Complex Systems ». In: *2019 IEEE Symposium Series on Computational Intelligence (SSCI)*. 2019 IEEE Symposium Series on Computational Intelligence (SSCI). Xiamen, China: IEEE, pp. 230–237. ISBN: 978-1-72812-485-8. DOI: 10.1109/SSCI44817.2019.9002840

— Hugo Cisneros, Josef Sivic, and Tomas Mikolov (July 1, 2020). « Visualizing Computation in Large-Scale Cellular Automata ». In: *Artificial Life Conference Proceedings*. ALife 2020. Vol. 32. MIT Press, pp. 239–247. DOI: 10.1162/isal_a_00277

— Hugo Cisneros, Tomas Mikolov, and Josef Sivic (Nov. 28, 2022). « Benchmarking Learning Efficiency in Deep Reservoir Computing ». In: *Proceedings of The 1st Conference on Lifelong Learning Agents*. Conference on Lifelong Learning Agents (CoLLAs 2022). Vol. 199. PMLR, pp. 532–547

The code to reproduce the experiments of all three publications is available on GitHub: https://github.com/hugcis/evolving-structures-in-complex-systems, https://github.com/hugcis/benchmark_learning_efficiency. We also published a dataset that we used for our benchmark in our last publication. It is available at https://github.com/hugcis/incremental_tasks/.

Chapter

2 Background

In this chapter, we provide background on some of the topics that we study in the thesis. We begin by describing the cellular automaton (CA) model, which is the complex system we focus on in most of the thesis (Section 2.1). Next, we describe the similarities between CAs and recurrent neural networks (RNNs), which is useful for understanding the characteristics and applications these two models share (Section 2.2). CAs can be completely reformulated as a special case of RNNs. This reformulation makes it natural to replace recurrent RNNs with CAs in several use cases. We describe the reservoir computing (RC) model (Section 2.3), which enables sending inputs to and harvesting the computations of complex systems to perform machine learning tasks without requiring explicit knowledge of the underlying system dynamics or the need to solve complex optimization problems. We apply reservoir computing to sequential language-like tasks in Chapter 6. This model was developed with RNNs, but we describe how CAs and other complex systems can be a suitable replacement, and the benefits of this approach.

2.1 Cellular automata

Stanislaw Ulam and John von Neumann proposed the CA model in the 1940s as a means of modeling crystal growth and creating an autonomous self-replicating system (Von Neumann and Burks 1966).

2.1.1 Definition

It is usually defined in a regular lattice in one or two dimensions. Each of its components is called a cell and can be in a state $k \in \mathcal{S}$. $\mathcal{S}$ is the space of available states for cells, usually chosen to be $\{0, 1\}$ for binary CAs or $\{1, \ldots, n\}$ for CAs with n states.

A neighborhood function $\boldsymbol{N}$ is defined that associates each cell with the indexes of its neighbors in the grid. In general, a CA can be constructed on any space $\mathcal{L}$ where this function can be defined. The space $\mathcal{L}$ specifies an index and the relationship between the cells. In practice, regular finite or infinite grids are chosen, $\mathcal{L} \subset \mathbb{Z}$ or $\mathcal{L} \subset \mathbb{Z}^2$. For example, the grid could be a one-dimensional torus with 10 cells, that is, $\mathcal{L}_{T_{10}} = \{1, 2, \ldots, 10\}$. The neighborhood function has the following general form:

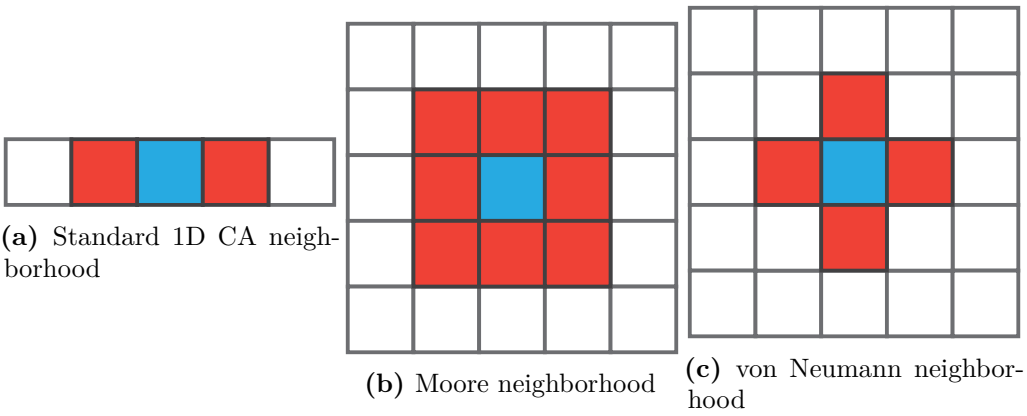
$$\begin{aligned} \boldsymbol{N}_{\mathcal{L}}: \quad & \mathcal{L} \to \mathcal{L}^s \\ & i \mapsto [c_j]_{j \in \mathcal{N}_{c_i}}, \end{aligned} \tag{2.1}$$

where $\mathcal{N}_{c_i}$ is the neighborhood of cell c_i, s is the number of cells in the neighborhood, and the returned value is a finite set of cells: the neighbors of cell c_i. For the torus $\mathcal{L}_{T_{10}}$ above, we can define the neighbors to be the cell itself and the two immediately adjacent cells. This type of 1D neighborhood is usually associated with a *radius* parameter $r = 1$. The neighborhood $r = 1$ is illustrated in Figure 2.1a. It corresponds to the following neighborhood function:

$$\begin{aligned} \boldsymbol{N}_{\mathcal{L}_{T_{10}}}: \quad & \mathcal{L} \to \mathcal{L}^3 \\ & \boldsymbol{N}_{\mathcal{L}_{T_{10}}}(i) = \begin{cases} [c_{i-1}, c_i, c_{i+1}], & \text{if} \quad i \in \{2, \ldots, 9\} \\ [c_{10}, c_1, c_2], & \text{if} \quad i = 1 \\ [c_9, c_{10}, c_1], & \text{if} \quad i = 10. \end{cases} \end{aligned} \tag{2.2}$$

A larger radius r would correspond to including more cells in the neighborhood, going in each direction from the initial cell. On two-dimensional grids, there are multiple ways to define the neighborhood. Some common examples are the Moore neighborhood (see Figure 2.1b) and the von Neumann neighborhood (see Figure 2.1c). In the rest, we omit the subscript on the neighborhood function $\boldsymbol{N}$ as we almost always work with regular grids on a torus, for which it is simpler to just mention the size and dimension.

A CA evolves in discrete time steps. An update rule $\boldsymbol{\Phi}: \mathcal{S}^s \to \mathcal{S}$ defines the new state of a cell as a function of its local neighborhood at the current time step. The local group of cells is said to be in a particular *configuration*. The function $\boldsymbol{\Phi}$ is applied in parallel to all cells. For a CA in its initial state at time step 0 — i.e., a set of cells



(a) Standard 1D CA neighborhood

(b) Moore neighborhood

(c) von Neumann neighborhood

Figure 2.1 – Illustration of commonly used neighborhoods for 1D and 2D CA.

$\left(c_i^{(0)}\right)_{i\in\mathcal{L}} \in \mathcal{S}^{|\mathcal{L}|}$, and a neighborhood function $\boldsymbol{N}$, we have the following update rule:

$$\forall i \in \mathcal{L}, \quad c_i^{(t+1)} = \boldsymbol{\Phi}\left(\left(c_j^{(t)}\right)_{j\in\boldsymbol{N}(c_i)}\right). \tag{2.3}$$

The details of an update step in a 1-dimensional CA is shown in Figure 2.2 for example. The neighborhood of the cell c_i is c_i itself, as well as its two immediately adjacent cells.

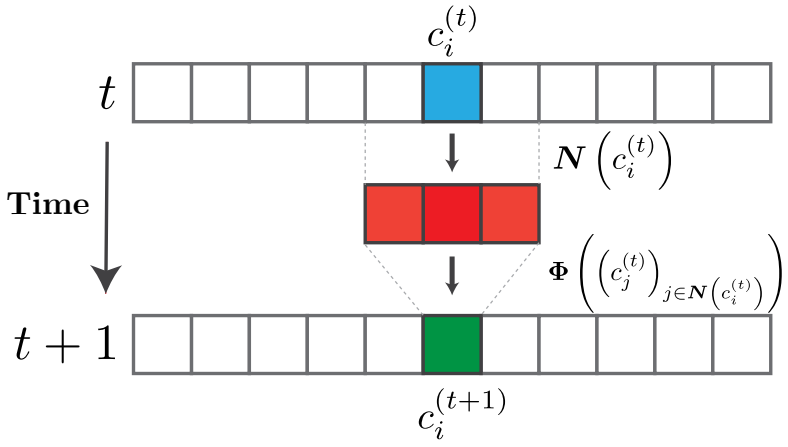


Figure 2.2 – Illustration of a cellular automaton update rule in 1 dimension. For each cell, we look up the neighboring cells and update their state according to the current state of the neighbors. This operation is applied in parallel to the entire grid.

The function $\boldsymbol{\Phi}$ is sometimes called a *rule table* because it associates an output state for each possible combination of input neighbor states. In an implementation of the CA model, these output states can be looked up from a table or array containing all possible transitions from input to output.

There are many ways to define CAs, and we gave a broad definition that includes many rarely used CA variants. For another more straightforward definition of CAs, see, for example, (Kari 2012).

Rule representation. A useful rule representation can be obtained by listing all output states corresponding to input neighborhood configurations in a predetermined order. This results in a list of values $[o_1, \ldots, o_{s^{|\mathcal{S}|}}]$, with $\forall i,\ o_i \in \mathcal{S}$, where s is the number of cells in a neighborhood, $\mathcal{S}$ is the space of available states, and $|\mathcal{S}|$ is the number of available states per cell. Using o_i as the digits of a base-$|\mathcal{S}|$ number, each rule is uniquely represented by a number. For example, as explained in more detail in Section 2.1.3, a one-dimensional cellular automaton with a neighborhood of size three has eight possible configurations of neighboring cells (000, 001, 010, 011, 100, 101, 110, and 111), each of which can map to two states (0 or 1). To obtain the rule number, we concatenate the new states for each of the input configurations, resulting in an 8-bit number. The 256 possible binary rules in one dimension with neighborhood size 3 can be numbered from 0 to 255 with this standard method and are referred to by their number in the literature.

Boundary conditions. The grid of a CA can be finite or infinite. In the infinite case, the grid is assumed to be initialized to a uniform state, except for a few cells set to other states. The simulation is then run on these few cells, while the rest of the infinite grid does not have to be simulated from the start. For a finite grid, an exhaustive simulation can be run, but one needs to define boundary conditions. The boundaries can be set to wrap around the other side of the grid, forming a torus. An example of the corresponding indexing is given in equation 2.2. Other choices of boundary conditions consist of adding virtual padding cells outside of the main grid. They can be set to a fixed state, a randomly chosen state, or they can mirror the cells inside the grid. Each of these choices affects the evolution and properties of the CA, but the importance of these boundaries decreases for very large grids.

2.1.2 Classification of cellular automata

Stephen Wolfram approached CAs as discrete, spatially extended dynamical systems (Wolfram 1984). The analogy is only superficial since many concepts from dynamical

systems theory, such as “chaos”, “attractors” and “sensitivity to initial conditions” only admit a rigorous definition in the continuous state and continuous time models. Wolfram proposed a qualitative classification of CA update rule behaviors roughly analogous to classifications in dynamical systems theory, with four classes defined as follows.

Class 1 All initial configurations converge to a single fixed configuration, such as a configuration consisting entirely of 1s.

Class 2 All initial configurations relax after a transient period to some fixed point or some temporally periodic cycle of configurations, but which one depends on the initial configuration. (CA defined on finite lattices always have periodic behavior because there is only a finite number of grid configurations. Class 2 does not refer to this type of periodic behavior but rather to cycles with periods much shorter than the total number of possible states).

Class 3 All initial configurations exhibit chaotic behavior after a short transient period. (The term “chaotic” here refers to apparently unpredictable space-time behavior, not the standard notion of chaos in dynamical system theory.)

Class 4 Some initial configurations result in complex localized structures that can persist for a long time.

This classification being qualitative and not rigorous, there is not even a clear consensus on which of the Elementary cellular automata (ECAs) rules (one dimensional binary CAs with neighborhood size 3) belong to class 4 or class 3. Class 4 CA rules are speculated to be capable of universal computation (Wolfram 1984), which refers to the ability of a system to simulate any other system or computation, given enough time and memory. For example, Li and Norman H. Packard (1990) claimed that ECA rule 110 has class 4 behavior and was eventually proven to be universal (Cook 2004), but no other ECA has been proven universal since.

Zenil (2010) studied the compression size of the space-time diagrams of all ECA for a fixed-length simulation. Using a simple k-means clustering technique, he obtained two groups that roughly match Wolfram’s classes 1 and 2 and classes 3 and 4. We reproduce these results in Figure 4.1b in Chapter 4. They offer an interesting nonqualitative confirmation of the results of Wolfram’s classification. However, these results vary signif-

icantly if we modify the initial conditions, as well as the grid size, data representation, or compression algorithm (Hudcová and Mikolov 2020).

Wuensche and Lesser (1992) studied the behavior of ECAs when the simulations are reversed and computed the preimages of each configuration. They introduced the Z-parameter, which is the probability that a partial preimage can be extended by one symbol for each CA. The authors speculate that class 4 occurs at $Z \approx 0.75$. This is not a classification but a class 4 membership test that can be computed from the rule directly, making it practical compared to alternatives. However, when tested in practice, Wuensche's hypothesis that $Z \approx 0.75$ corresponds to class 4 is rarely verified.

Hudcová defined a classification of CA that is based on the form of the asymptotic growth of its transients, obtained by repeated simulation of the rules of CA starting from various initial conditions and using the best fitting function for the asymptotic growth of its transients (Hudcová and Mikolov 2020, 2022). For a detailed review of the current methods and challenges of CAs classifications, see (Vispoel, Daly, and Baetens 2022).

2.1.3 Cellular automata variants

Several CA variants have been proposed, modifying or restricting various parts of the definition in Section 2.1.1. Here, we list some common ones.

Elementary cellular automata. ECAs are 1 dimensional CAs with two states per cell and neighborhood size 3 — the cell and its two direct neighbors on the 1D grid. There are 8 possible configurations of a neighborhood with 3 cells and 2 states per cell, which corresponds to 256 possible ways to define an ECA — two possible outputs for each of these 8 possible configurations; hence $2^8 = 256$ possible CA. This relatively small number of rules allows for an exhaustive exploration of the rule space and mapping of the properties of ECA, which would not be possible for general CAs.

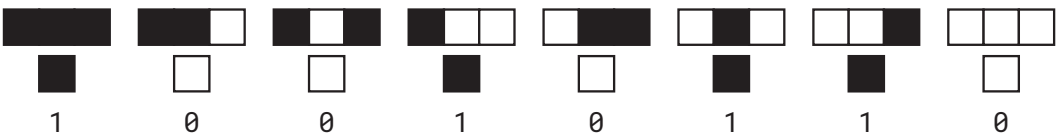


Figure 2.3 – Illustration of the rule of ECA number 150.

These CAs have been extensively studied and offer an interesting combination of

trivial definition and implementation, and complex and unpredictable properties. One of the fundamental problems of CA research is to classify the 256 rules into well-defined behavior types and order them by complexity, which was attempted in several previous works (H. A. Gutowitz 1991; Hudcová and Mikolov 2020, 2021; Wolfram 2002; Wuensche 1999; Wuensche and Lesser 1992; Zenil 2010). We discuss these classifications in more detail in Section 2.1.2.

(a) Single cell initialization **(b)** Random initialization

Figure 2.4 – Evolution of ECA number 150 simulated on a grid of size 100 for 50 steps. 2.4a shows a simulation starting from a blank grid with only one cell set to 1. 2.4b shows a simulation starting from a random initialization. Time flows from top to bottom, and each row of cells represents a single internal CA state at a single time step.

ECA rules are easily visualized because there are only eight possible neighborhood configurations that need to be shown. For example, Figure 2.3 shows the transition table for ECA Rule 150. If we assign 1 to the black state and 0 to the white state, the possible neighborhood configurations are ordered in their binary order from right to left, with the leftmost bit being the most significant. This is how the rule number is computed, using the output states (last row in Figure 2.3) as a binary representation. Example simulations of ECA rule 150 starting from various initial states are shown in Figure 2.4. In these representations, time flows from top to bottom, and each row of cells represents a single internal CA state at a single time step. This representation of an ECA rule is often called a *spacetime diagram*.

There are several well-known ECA rules with particularly interesting properties. For example, a universal computer has been constructed in rule 110 (Cook 2004), making it the first (and only to date, although ECA Rule 54 is postulated to be as well) Turing complete ECA.

Totalistic cellular automata. Totalistic CA are a subset of CA whose rules can be expressed as a function of the sum of neighboring cell values. These CA were introduced by Stephen Wolfram (Wolfram 1983). ECA 150 is an example of a totalistic CA, and its rule and typical space-time diagram are depicted in Figures 2.3 and 2.4. Its rule could be summarized in natural language as "if exactly two states are 1 or all states are 0, the next state is 0. Otherwise, if one or three states are 1, the next state is 1". Conway's Game of Life, presented in the following paragraph, is an example of a totalistic CA in two dimensions.

Game of Life. The game of life is one of the most famous CA. It was proposed by the mathematician John Conway in 1970 (Gardner 1970). It is a two-dimensional binary CA. Its two states are often called "alive" and "dead". Its rule can be summarized in three sentences as follows.

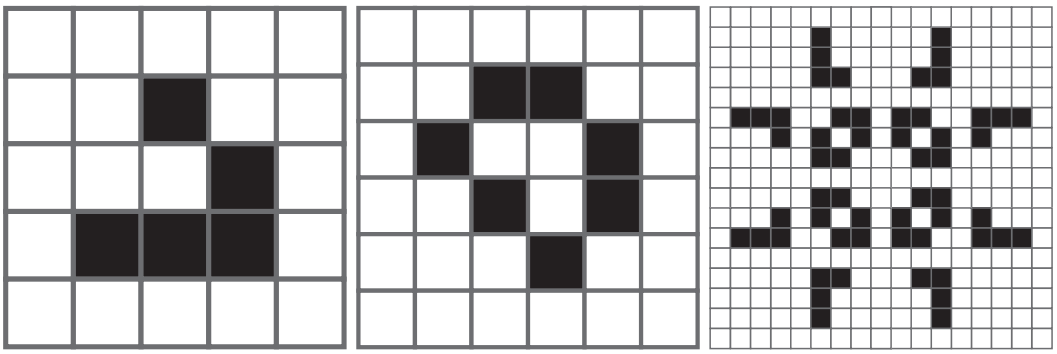
— Any live cell with two or three live neighbors survives.

— Any dead cell with three living neighbors becomes a live cell.

— Other live cells and already dead cells are dead in the next generation.

This CA has a particularly active community dedicated to finding interesting patterns with particular properties, such as long cycling periods or particular speeds of movement through the grid. Some of these patterns have been used as memory registers and communication channels to build a Universal Computer (*Igblan - Life Universal Computer* 2022). This construction proved that the game of life is Turing complete. This property is thought to be also shared by other CAs — it is proven for at least ECA rule 110 (Cook 2004). This makes CAs models theoretically appealing for the design of a learning algorithm because they can simulate any algorithm.

Asynchronous cellular automata. A cellular automaton is said to be asynchronous when its cells are not updated in parallel at each time step. Various cell update schemes can be chosen. Asynchronous CAs have to define an order of update of the cells, which is not easily defined on an infinite grid. Therefore, most implementations are done on finite grids. This update order can be random, follow a predefined order, or be decided by a global controller. Groups of cells can be updated simultaneously, or updates can be done cell by cell. The many effects of asynchronous updates can be difficult to predict



(a) Moving oscillator (*glider*).

(b) Fixed pattern (*still life*).

(c) Period 3 oscillator (*pulsar*).

Figure 2.5 – Some game of life patterns with various properties. The moving oscillator 2.5a moves 1 cell along the bottom right diagonal every 4 steps. The fixed pattern 2.5b never changes except if it interacts with others. The oscillator 2.5c loops through three different configurations.

and can make these types of CAs relatively difficult to work with.

Asynchronous CAs are an attractive model when using large systems for which parallel update is prohibitively expensive. The rule could adapt itself and preferentially update cells in active or useful parts of the CA while doing slower updates to the less active parts of the CA. An asynchronous CA with evolutionary properties was constructed in (Nehaniv 2003). The authors argue that asynchronicity is a strong advantage in the construction of systems that can evolve within CAs.

Stochastic cellular automata. In a stochastic cellular automaton, the update function $\mathbf{\Phi}$ is stochastic. This means that the next state of all cells in the grid is sampled from a probability distribution that depends on the current neighborhood configuration. We can write

$$\forall i \in \mathcal{L},\ s \in \mathcal{S} \quad p\left(c_i^{(t+1)} = s\right) = \mathbf{\Phi}\left(\boldsymbol{N}\left(c_i^{(t)}\right)\right)(s). \tag{2.4}$$

These CAs have had successful applications in physics (Ottavi and Parodi 1989; Vichniac 1984) or biology (Boas et al. 2018), because their stochasticity allows modeling complex physical phenomena.

Continuous cellular automata. Another family of CAs uses real-valued states, often restricted to the range $[0, 1]$. The update function is then a real multivariate function,

which we can write

$$\mathbf{\Phi} : [0,1]^s \to [0,1]. \tag{2.5}$$

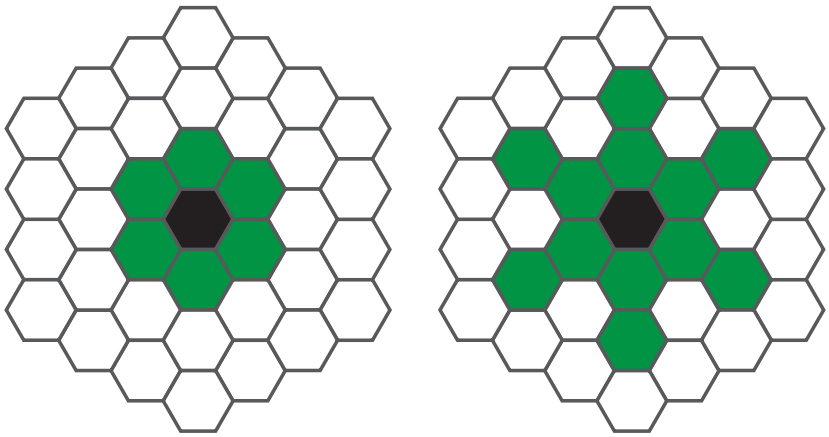
Examples of continuous CAs include Lenia, which uses convolutional operators followed by thresholding for the update rule (Chan 2019), or neural cellular automata (NCAs) (Mordvintsev et al. 2020) which represent CAs as neural networks (more details in Sections 2.2 and 3.5.1). Garzon and Botelho (1993) explores the condition for a real-valued CA to be able to compute real-valued functions.

Higher dimensional cellular automata. CAs of dimension greater than 2 have been comparatively less studied for several reasons, including the limits that arise when simulating and visualizing systems in more than 2 dimensions on a computer screen. Another problem is with the increasing number of possible rules in higher dimensions. The number of neighbors per cell grows exponentially with dimension, and the number of possible rules has a doubly exponential growth rate. This makes the convenient representation of CAs rules as tables infeasible since the size of that table quickly exceeds the available memory of most computers. For example, there are $2^9 = 512$ possible 3×3 configurations of a binary Moore neighborhood (see Figure 2.1b) in 2D that can each lead to 2 states, which means that there exists $2^{512} \approx 10^{154}$ distinct 2D rules with 2 states and a Moore neighborhood. This is already an incomprehensibly large number, but there are $2^{134,217,728} \approx 10^{40,403,562}$ such rules in 3D. In general, there are 2^{3^n} possible binary CA rules with a Moore-like neighborhood in dimension n.

Despite these limitations, several works have studied higher dimensions CAs, although mostly in 3 dimensions (Sudhakaran et al. 2021; Tsalides, Hicks, and York 1989). For example, there are several examples of successful 3D CA simulations applied to material sciences (Arata et al. 1999; Di Caprio et al. 2016; Gandin and Rappaz 1997; Pan, Feng, and Hudson 2009).

Hexagonal cellular automata. Hexagonal CAs are defined on grids tiled with hexagons, where each cell has 6 direct neighbors.

One advantage of this type of CAs over regular grids is the absence of preferred direction since they are all equivalent. In a square grid, diagonally adjacent cells span



(a) Range 1 neighborhood **(b)** Ray neighborhood

Figure 2.6 – Example definition of the neighborhood for hexagonal cellular automata.

more distance than directly adjacent cells, whereas on a hexagonal grid, all neighbors are equivalent (Moree and Riele 2004).

2.1.4 Parametrizing and sampling CA rules

As explained in Section 1.3, one of the main challenges of working with complex systems is the parametrization and control of the behavior of that system. CAs are no exception, and several parametrization schemes have been proposed, with their respective benefits and drawbacks. There are infinitely many ways to define rules for CAs, and for each of these definitions, a very large number of rules are available, making it impossible to sample a significant portion of the rule space. A good parametrization allows one to scan a wide range of behaviors of CAs by traversing the space in an interesting way.

Langton's lambda. Christopher Langton proposed one of the first parameters that could be used to control the behavior of CAs to a certain extent (Langton 1986, 1990). For a CA with $K = |\mathcal{S}|$ states and a state arbitrarily chosen to be the *quiescent* state (this corresponds to the "dead" state in the game of life, for example), if there are n transitions to the quiescent state within the rule table and $K^s - n$ remaining transitions that do not lead to the quiescent state, where s is the number of cells in the neighborhood, we have

$$\lambda = \frac{K^s - n}{K^s}. \tag{2.6}$$

For Langton, the purpose of λ is to search the space of CAs in an ordered manner by varying the value of the parameter. His procedure starts with a rule with all transitions leading to the quiescent state. The value of λ is increased in discrete steps to $1 - 1/K^s$ by randomly changing the transitions of the rule to lead to a different state. An illustration of the behavior change of a CA rule under this procedure is shown in Figure 2.7. Langton collected various measures of the dynamical behavior of CA and studied them as a function of λ. He observes a phase transition as λ approaches its maximal value, where CAs seem to behave in the most complex way, with long transients and large-scale propagating structures. He calls this phase transition "edge of chaos" because it corresponds to λ values just before the generated CAs become chaotic.

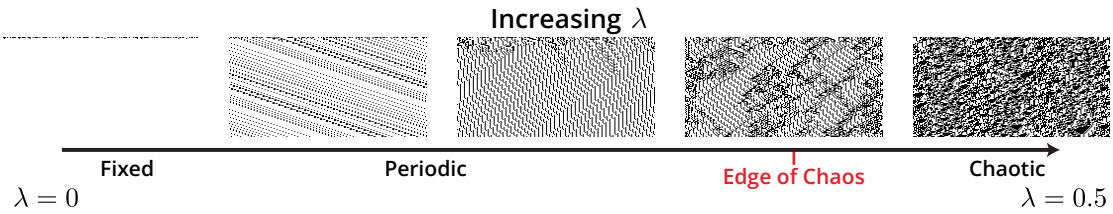


Figure 2.7 – The effect of varying λ for a 1D binary CA rule. A CA rule is progressively modified along a trajectory of increasing λ. The rule starts with a fixed behavior, becomes periodic, and then complex when hitting the "edge of chaos". Note the localized structures that spread over time (down along the y axis) and interact in the fourth figure. When λ increases further the rule becomes chaotic.

Dirichlet sampling. For an arbitrary number of states and an arbitrary neighborhood size, it can be challenging to sample CAs while scanning the wide range of behavior these models are capable of simulating. This is because the size of the space of CAs rules is very large. The naive method of uniformly sampling each transition output is not useful in large rule spaces with multiple states, large neighborhoods, or grid dimensions larger than 1.

Rules with equal proportions of transitions leading to all states tend to be the most chaotic. This is also what Langton observed when studying the λ parameter. This is explained by the fact that sampling each transition output uniformly is equivalent to sampling the rules from a multinomial distribution with the number of possible states (K), number of possible transitions ($n = K^{K^s}$, with s the number of cells in a neighborhood), and the uniform probabilities $p_1 = \ldots = p_K$ as parameters. A random

variable $X = (X_1, \ldots, X_K)$ indicates the number of times each result is observed. The probability mass function of this distribution is

$$P(X_1 = x_1, \ldots, X_K = x_k) = \frac{n!}{x_1! \cdots x_K!} p_1^{x_1} \cdots p_K^{x_K}. \tag{2.7}$$

This quantity becomes vanishingly small for large n (that is, rules with many states or large neighborhoods) and rules with an output transition skewed towards a specific state. For example, the binary Game of Life rule has 372 transitions leading to state 0 and 140 leading to state 1. The probability of sampling a rule with these transition proportions is 8.24e−26 (compared to a 17% probability of sampling a rule with between 254 and 257 transitions to 0). Using a uniform transition sampling method makes it highly improbable to obtain a rule resembling the Game of Life, let alone the rule itself.

We propose a Dirichlet sampling method as an alternative way to sample rules with fixed ratios of transitions leading to each output state. For a rule with K states, we sample a K-uple from a Dirichlet distribution of order K with parameters $(\alpha_k)_{k\in[1,K]}$ where $\alpha_0 = \alpha_1 = \ldots = \alpha_K = \alpha < 1$. The result is a quantile K-uple $(q_1, \ldots, q_K)$. For $\alpha < 1$, the distribution is concentrated around the corners of a simplex of dimension K, which means that it preferably samples a tuple with one of its values dominating the others.

Rules are sampled so that the number of transitions to each output state matches the quantile generated from the Dirichlet distribution. The samples will be more likely to have a dominant quantile, which can be associated with the quiescent state in Langton's λ calculation. The resulting sampling of cellular automata is much better for scanning the entire space of rules and generating CA rules that would be impossible to reach with the naive sampling method, as illustrated in Figure 2.8.

Smooth sampling with the recurrent convolutional neural network analogy. The recurrent convolutional network analogy of CAs (see Section 2.2 for details) formulates a CA as a neural network. In this paradigm, the rule of a CA is completely determined by the parameters of that neural network. This allows us to modify the CA rules as we would change the weights of a neural network. For example, as long as all simulation steps are differentiable, the backpropagation algorithm can be applied and

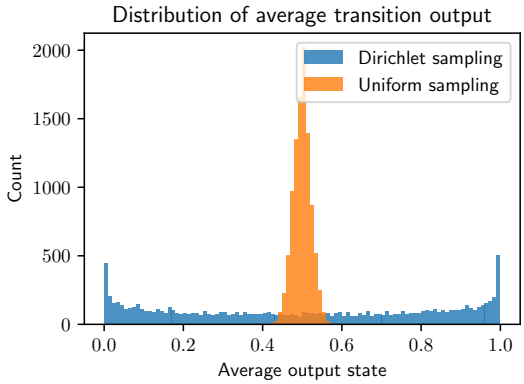


Figure 2.8 – An illustration of the difference between naive uniform rule sampling and Dirichlet sampling on binary rules. 10000 2D binary CA rules were sampled with each method, the plot displays a histogram of their average output state. Uniform sampling leads to more rules with an average output state close to 0.5, meaning approximately as many transitions toward both states. Dirichlet sampling can control, through the choice of parameter α, how many rules with more skewed transitions should be sampled, which is shown by the spikes on the blue histogram close to 0 and 1.

used to update a CA rule by updating the weights of the neural network with gradient descent. Dimensionality reduction methods applied to the parameter space of the neural network can help to understand the structure of that space and the CAs behaviors produced by different sets of parameters.

2.2 Cellular automata and Recurrent neural networks

The purpose of this section is to show that cellular automata and recurrent convolutional neural networks have very strong connections and to draw this parallel as clearly as possible. We express the CA update function as a set of convolutional operations that can be derived from any CA rule.

This connection yields interesting consequences for both the theoretical properties of these models and the potential applications of CAs and recurrent networks. It shows, for instance, that emergent properties with increasing complexity and perhaps open-ended development that we expect from complex systems such as CAs are possible within the hidden state of a recurrent neural network.

The conceptual similarity between CAs and neural networks is not surprising. Fundamentally, CAs represent the paradigm of emergence, in which the origins of complex

behaviors are searched for in an assemblage of possibly very simple parts rather than being viewed as a sum of complex building blocks. It is often argued that there is no better-known example of a truly emergent phenomenon than that of the emergence of consciousness out of the large network of functionally simple (and certainly unconscious) neuronal components that make up the human brain. A biological neural network consists of a large space of interconnected nodes with dynamic behavior that is a local function of the other nodes to which it is connected, which is strikingly similar to the principle of a CA. Artificial neural networks can be thought of as being a set of biologically inspired CA rules (Ilachinski 2001). The more precise parallel between cellular automata and a form of recurrent/convolutional network has also been drawn by several other researchers (Gilpin 2018; Mordvintsev et al. 2020; Wulff and Hertz 1993).

In this section, we formally describe the relation between CAs and RNNs and discuss the extensions this implies for the CA model, as well as some of the consequences of that relation.

2.2.1 RNN formalism

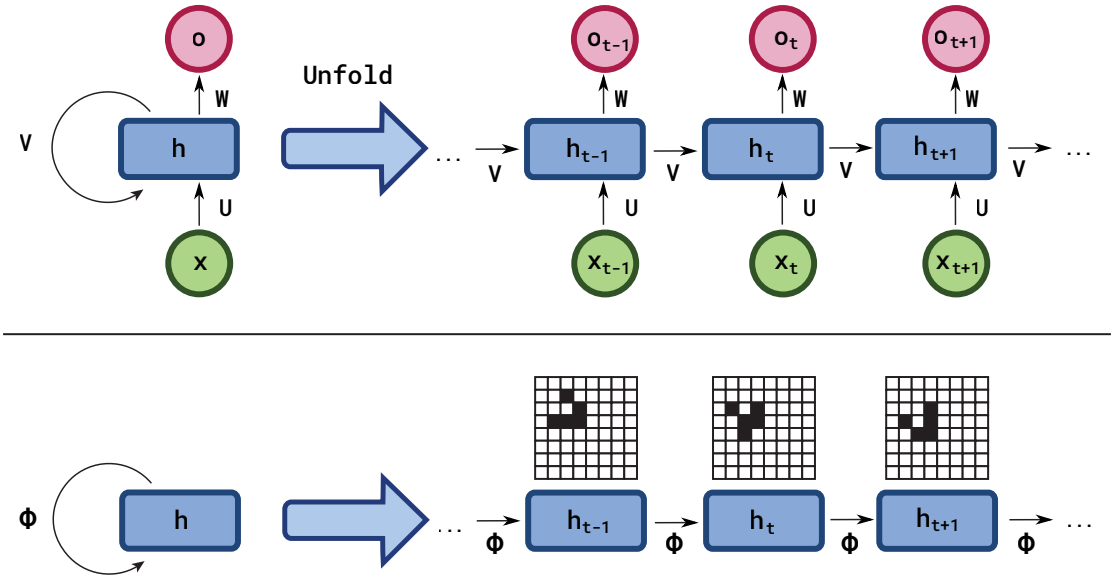


Figure 2.9 – Standard RNN architecture (top) and Game of Life seen as a RNN with no inputs and outputs (bottom). For the CA, the "hidden" state h_t is the current state of the grid at timestep t. The operator $\mathbf{\Phi}$ is the CA update rule which is equivalent to a convolutional neural network (CNN) (see Section 2.2.2). This illustration is based on Recurrent neural network unfold by fdeloche, licensed under CC BY 4.0.

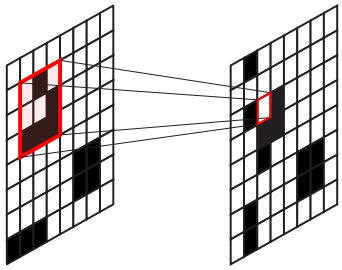


Figure 2.10 – The CA update rule is local and can be represented by a set of CNNs: linear local uniform transformations followed by the application of a non-linear function.

We write the definition of a CA with a formalism and notation inspired by RNNs. This parallel is illustrated in Figures 2.9 and 2.10. The state of the grid at time t is denoted h_t and corresponds to the hidden state in a RNN. In the case of classical CAs, it is a 1 or 2D vector of discrete values (see Section 2.1.1), but (Mordvintsev et al. 2020) and other CAs extensions use continuous values, much like usual RNNs.

The mapping Φ operates only on the hidden state. Because it is local, it can be shown to be equivalent to a convolutional layer, as explained below in Section 2.2.2. The inputs and outputs $(\mathbf{x}, \mathbf{o})$ in Figure 2.9 are not included in the classical definition of CAs, but are an easy-to-implement extension, as discussed in Section 2.2.3.

2.2.2 Transition rule as a set of convolutions

Each cell c_i from the CA definition can be in one of the k states. We represent the cells as vectors of size k, where k is the number of states, with only zeros excepted for a one in the k-th position, which can be interpreted as a one-hot encoding of the cell state. A neighborhood of size 3 in a 1D CA can be represented as a $3 \times k$ vector $\mathbf{u}_i = [u_{i-1}, u_i, u_{i+1}]$. Each u_i is a vector of size k with a 1 in position s_i. This is illustrated in the 1D (resp. 2D) case on the left (resp. right) of Figure 2.11. For a CA with only two states, it is redundant to have a 3×2 vector, but the "one-hot" encoding becomes useful when working with more states.

With this representation, we can express the transition rule Φ as a simple convolutional neural network. This network would be made up of two layers, which are shown in Figure 2.12 for the 2D case. We describe a simple construction of a CNN representing any binary CA rule below:

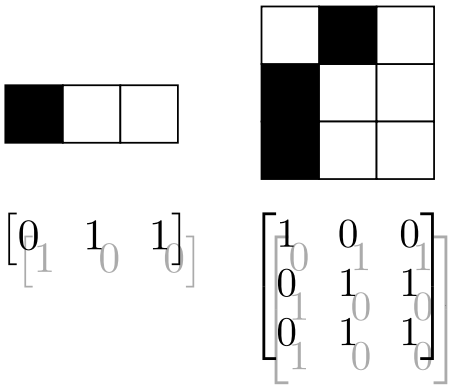
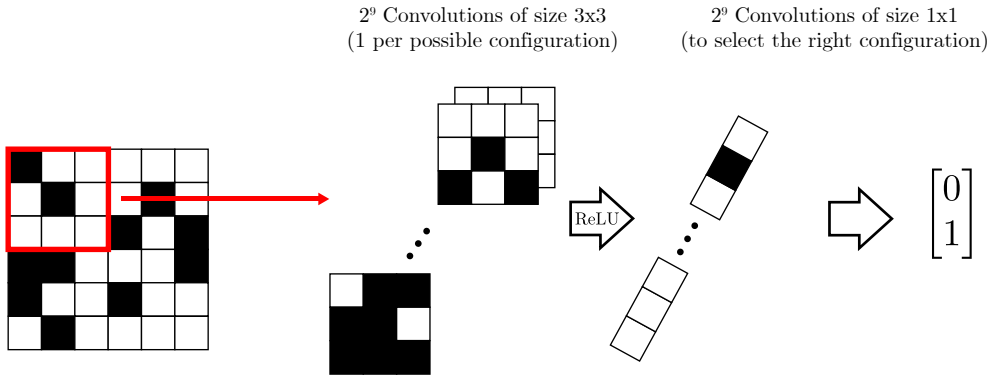
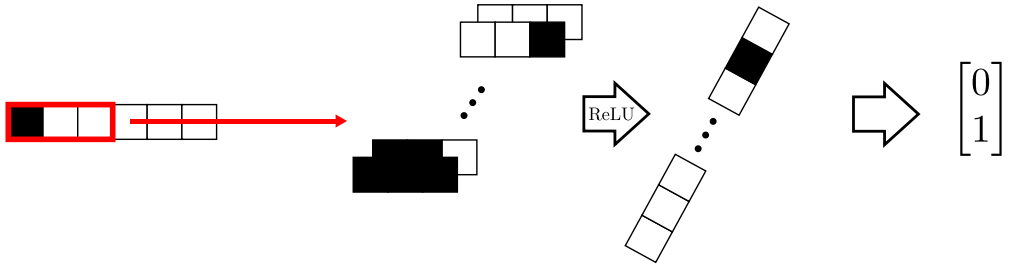


Figure 2.11 – A CA neighborhood vector representation example with 2 states. A 3×3 square of cells with two states can be represented by a $3 \times 3 \times 2$ tensor. Left: 1D 3-neighbors representation. Right: 2D 3×3 neighbors representation. The last dimension of the vector encodes a "one-hot" representation of the state of the cell.



(a) A representation of any binary 2D CA rule operating on 3×3 neighborhoods as a 2-layer CNN. There are 512 filters receptive to each 3×3 neighborhood states in the first layer. The second layer determines the output value for each neighborhood state.



(b) A representation of any binary 1D CA rule operating on neighborhoods of size 3 as a 2-layer CNN. There are 8 filters of size 3×3 in the first convolutional layer, and 8 filters of size 1×1 in the second one.

Figure 2.12 – Two examples of CA rules represented as a simple CNN. Padding ensures the grid obtained after the CNN step is of the same size as the original grid.

The first convolutional layer. It is receptive to each possible neighborhood configuration. In the 1D case with neighborhood size 3, this is all eight configurations $000, 001, 010, \ldots, 111$. For a radius r in 1D, this layer is composed of k^{2r+1} filters of dimension $k \times (2r+1)$ with only ones and zeros, which are the mirror of a possible configuration of size $(2r+1)$ of the neighborhood.

The product of each filter with an input neighborhood in the grid will be an integer between 0 and $2r+1$. Applying a ReLU nonlinearity preceded by a constant vector of value $2r$ as bias, we obtain the input to the second layer, a vector of size k^{2r+1} for each cell containing all zeros except for one corresponding to the detected neighborhood. This vector plays the role of an indicator of the current input configuration.

A second convolutional layer. It has k^{2r+1} filters of size 1 that are 0s or 1s depending on the desired output value of that transition. When applied to the indicator vector above, the output obtained will be the output value of the rule corresponding to the detected neighborhood state.

With some circular padding that wraps around the edges of the grid or applies constant values outside of the main grid, the size of the output is the same as the input. This means that the CNN step can be applied multiple times, simulating a single CA step every time.

2.2.3 Adding inputs and outputs

Figure 2.9 shows the locations where inputs and outputs can be incorporated into the Game of Life or any other CA. We provide a variety of examples of how input and output can be integrated and how they can interact with the rule of the CA.

Inputs

There are many ways to add inputs to the recurrent CNN model presented above. We divide our proposed methods into two classes: inputs that directly modify the hidden state during the CA evolution and inputs that augment the hidden state without changing it directly and affect the rule.

Hidden-state augmentation and rule modulation. Inputs are usually added to the hidden state just before the application of the non-linearity in standard RNNs.

In the case of cellular automata, it can also be done in the following ways:

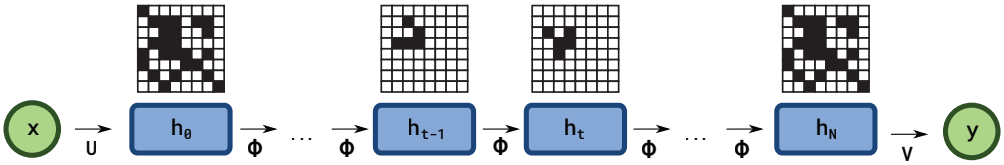


Figure 2.13 – Inputs x and outputs y can be directly encoded within the hidden state. U and V are the operators mapping x to a hidden state tensor and decoding a hidden state tensor to output y.

— **Vector state.** A straightforward way to add inputs to a CA is to consider the grid as having an additional (possibly read-only) dimension. For example, a 1D automaton of size N would be represented by a vector of dimension $N \times 2$ where the first vector of size N is the state of the grid and the other is the input. The update rule would be changed to take into account this new component. One can see this model as using multiple CA rules at the same time, with the input state conditioning the rule chosen for a given update step.

— **Variable update rule.** Inputs can also influence a CA's evolution by changing the update rule. With this configuration, the update rule Φ is now a function of the input x. We now have $\Phi_x = G(x)$, where

$$G : \mathcal{X} \to \left(\{1, \dots, k\}^{2r+1} \to \{1, \dots, k\} \right),$$

and $\mathcal{X}$ is the input space.

A similar approach was used in (Adams et al. 2017). It showed that conditioning a CA update rule on another CA's evolution could enable a higher diversity of behavior.

— **Concatenation.** One way to incorporate input is by concatenating it with the hidden state, such as at the boundaries of the grid, before applying the update rule. This creates a fixed read-only memory space, as shown in Figure 2.14. However, a potential drawback of this method is that information must propagate through the grid in order to be processed in locations far from the read-only memory's position.

This approach is more reminiscent of computer architecture, where designated areas are reserved for specific data functions.

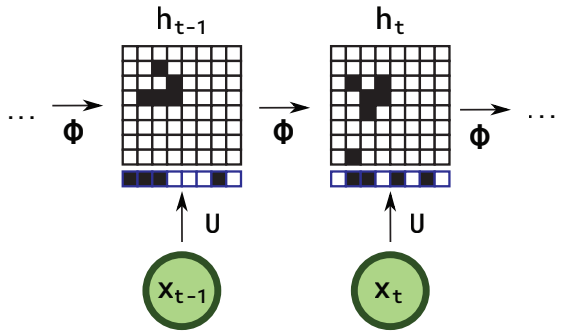


Figure 2.14 – Input x_t is projected and concatenated to hidden state h_t, affecting the boundary conditions of the CA. For the CA, this vector is just like another part of the hidden state except that it is not writable

Hidden-state manipulation. Another approach is to directly modify the hidden state to "communicate" information to the system. These methods are used in (Mordvintsev et al. 2020; Randazzo et al. 2020) to make interactive demos and allow users to directly modify that hidden state. A user can interact with the system by drawing with its mouse. This sets parts of the internal state to a fixed cell state.

— **Masking.** Input data can serve as a mask on top of the current grid state, forcing some cells into states that depend on the input values. If we view the state activations in the CA as some neural pattern, this approach can be seen as a form of *neuromodulation*, which has previously been used in machine learning (Beaulieu et al. 2020; Ishiguro, Fujii, and Hotz 2003; Soltoggio et al. 2008).

A mask can be constructed from an input vector x by linearly transforming x and applying an activation function (e.g., step function). It controls which channels are activated and where information can flow or not. This kind of mask can be applied with element-wise multiplication or addition but also more elaborate operations. It is illustrated in Figure 2.15.

Outputs

Similarly, inputs can be extracted from the hidden state in several different ways. One might read outputs from the boundaries of the grid, from a transformation of the

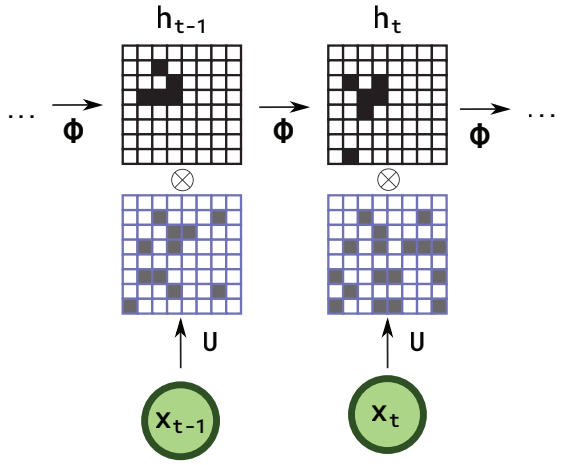


Figure 2.15 – Input is converted into a binary mask for the hidden state. The mask is applied through element-wise multiplication here.

grid using a neural network, etc. The output could also be stored within an additional dimension of the "extended" grid state presented above.

Cellular automata and computations

From a computational point of view, the hidden state can be seen as a working tape on which some kind of *parallel Turing machine* (the cellular automaton) performs computations. In this framework, the input is encoded as the initial state of the tape and decoded from the last state of the tape (as illustrated in Figure 2.13). The recurrent convolutional neural network (RCNN), or CA, then plays the role of a fixed computer program that is executed. This is the setting adopted in previous work on constructing computations with cellular automata: a task is chosen, and rules that can execute that task on input/output pairs are searched (Melanie Mitchell 2005).

Another interesting point of view can be taken when we consider the hidden state (or tape) as encoding both some data and a computer program, as is the case with a Turing machine. We can then expect a CA (or RNN) to not only compute the result of a particular function but to be equivalent to a general-purpose computer capable of computing the result of any chosen algorithm without any need for optimization. Making a universal CA compute some program amounts to encoding the program in the right language and communicating it by embedding it within the hidden state. In practice, figuring out the right encoding is often the biggest challenge. This is also the reason that the few Turing complete CAs are not so useful in practice since we do not know of

any efficient encoding for programming them.

2.2.4 Consequences

Viewing cellular automata as recurrent convolutional neural networks, as described above, has several interesting consequences, of which we list a few here.

Turing-completeness of the system

Because we can simulate rule 110 ECA in the above RCNN system, it follows from the Turing completeness of this CA rule that the RCNN is Turing complete. This is an interesting result, although not very significant since RNNs have already been proven to be Turing complete (Siegelmann and Sontag 1992), with very few practical implications. Our model is relatively far from a real-world CNN with a fixed number of layers independent from one another — compared to a variable number of steps and shared layers for the automaton-RCNN. Thus, this property does not seem to bear any consequence on CNNs.

Differentiable cellular automata

Each step of the computations involved in computing a step of a cellular automaton represented as a RCNN is differentiable. Therefore, we can theoretically couple this framework with backpropagation to create *learnable* cellular automata that can adapt their rules to minimize a target loss function.

This is the direction taken by Mordvintsev et al. (2020). The authors use supervised learning on a CA and train it to obtain a stable self-repairing target shape. However, because supervised CA can only do as much as they have been trained to do, it may defeat the purpose of working with a model capable of spontaneous complex emergent behavior. An open-ended complexity increase could very unlikely be achieved through pure supervision.

Gilpin (2018) takes the reverse approach and tries to train RCNNs to simulate a fixed CA rule, using the statistics of that training process as a way to help understand the structure of the CA rule space.

2.2.5 Beyond the naive rule representation

The representation of CA rules that we built in Section 2.2.2 is a one-to-one mapping. Each rule has its RCNN counterpart, and vice versa. However, there are several ways to make that representation of CA rules more efficient, using a neural network with fewer parameters. For example, we can make the first layer of filters receptive to both a given configuration and its inverse (e.g., $(1, 0, 1)$ and $(0, 1, 0)$ in an elementary cellular automaton) by using negative values in the filter (use a filter $(1, -1, 1)$ instead of $(1, 0, 1)$ and $(0, 1, 0)$ separately).

For example, the rule of the game of life does not need more than two convolutional filters on the first layer to be represented by a CNN because it is totalistic. The new state of a cell depends only on its number of neighbors in the alive state and the current state of the cell.

2.3 Reservoir computing

Reservoir computing (RC) is a computational framework that aims to exploit the states of a complex dynamical system to perform some target task (G. Tanaka et al. 2019). It relies on a *reservoir* of computations, that is, a dynamical system performing some computations on its internal state. An input signal is fed into this reservoir that behaves as a black box from the point of view of the algorithm. Input values can be sequential or nonsequential. They are projected and combined with the dynamical system using a suitable mechanism. This projection can be learned, set randomly, or chosen. The internal state of the dynamical system evolves according to its update rule, which is defined by the chosen system. Finally, a decoder model (usually a linear regression) is trained in a supervised manner to extract necessary information from the internal state of the dynamical systems to predict the right output. A general diagram describing this process is shown in figure 2.16.

Some early known formulations of the RC idea were done by Kirby (1991) and Schomaker (1990, 1991, 1992). In another early work, Buonomano and Merzenich (1995) used a random spiking neural network with excitatory and inhibitory elements. The probabilities of neuron connection was inspired by the real rat brain data (Mason, Nicoll,

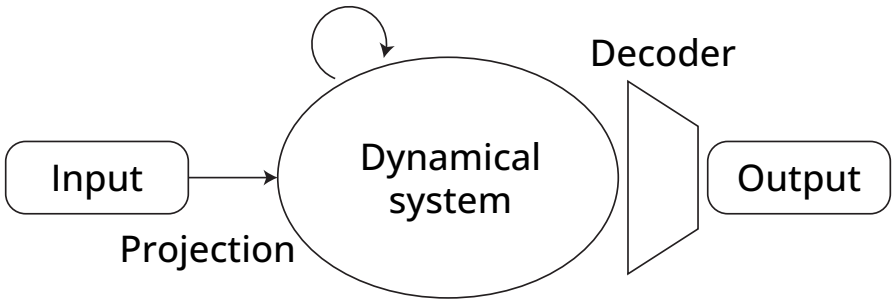


Figure 2.16 – Diagram of the general functioning of a reservoir computing (RC) system. Input values are projected in the state or evolution function of a dynamical system which is run according to its internal rule. Output values are decoded from the internal state of the dynamical systems with a trainable layer.

and Stratford 1991). The authors observed an interval-sensitive response of these neurons when subjected to pulses spaced by varying time intervals. This indicates an ability to encode temporal information. To measure this phenomenon, they trained an output linear layer to recognize specific patterns in the activation of the last layer of neurons. This idea of using a randomly initialized RNN with fixed weights and training a simple linear layer to decode outputs was reinvented and popularized later under the names "Echo-state networks" (Jaeger 2001) and "Liquid state machines" (Maass, Natschläger, and Markram 2002).

Another advantage of reservoir computing is that reservoirs can be implemented using a variety of physical systems, substrates, and devices. These types of reservoir are sometimes called "exotic" (Lukoševičius and Jaeger 2009) in contrast to RNN-based reservoirs. Physical RC is a promising area of research that may help us develop cheap and efficient machine-learning hardware and can inform us about the nature and behavior of the underlying physical systems. G. Tanaka et al. (2019) identifies four characteristics that make a reservoir suitable for solving a computational task.

1. The inputs must be mapped to a high-dimensional space, which corresponds to the internal state of the reservoir. This high-dimensionality ensures that the inputs are separated with high probability.
2. The reservoir should be non-linear. This will allow inputs that are not linearly separable to be separated as they are projected within the reservoir.
3. The reservoir should have the echo-state property (Jaeger 2001; Yildiz, Jaeger, and Kiebel 2012), or fading memory (Boyd and Chua 1985; Maass, Natschläger,

and Markram 2002, 2004). This property relates the asymptotic properties of the excited reservoir dynamics to the driving signal. A reservoir with the echo-state property will asymptotically forget all input values. This makes some reservoirs unable to solve tasks where unbounded memory is possibly required, such as context-free grammar parsing (Schmidhuber, Wierstra, et al. 2007). Later, reservoir models that address this issue were developed (Pascanu and Jaeger 2011).

4. A reservoir should separate responses from different signals into different parts of the space while being insensitive to small perturbations of the input. For a given system, this trade-off is often obtained at the transition between chaotic and non-chaotic regimes (Bertschinger and Natschläger 2004; Legenstein and Maass 2007).

We list common types of RC that are based on various families of reservoirs below:

Dynamical systems. This refers to a broad category of RC models based on other dynamical systems than the random RNNs. This includes reservoir computing with Cellular automata, which we give more detail about in section 2.3.3.

Electronic RC. These models are implemented on electronic circuits, such as artificial neural networks implemented on physical circuits or other neuromorphic circuits. For example, RC models can be build on FPGAs (Alomar et al. 2014; Antonik 2018; Antonik, Smerieri, et al. 2015; David Verstraeten, Benjamin Schrauwen, and Stroobandt 2005) or memristive circuits (Donahue et al. 2015; Merkel et al. 2014; Yang, Chen, and F. Z. Wang 2016).

Mechanical RC. Soft mechanical bodies have complex nonlinear dynamics that can be used as a reservoir to build a RC model (Pfeifer, Bongard, and Grand 2007). For example, a reservoir using a mass-spring network was proposed by Hauser et al. (2011).

Biological RC. There have been multiple hypotheses on whether the parts of the brain behave like a RC system (Yamazaki and S. Tanaka 2007). The goal of this area of research is to study the RC-like properties of biological systems rather than build reservoir computers. For example, a RC model was proposed to understand the mechanism of context-dependent eye movement (P. Dominey,

Arbib, and Joseph 1995; P. F. Dominey 1995).

Photonic RC. The principle of photonic RC is to use optical reservoirs to generate the computations. A first example using semiconductor optical amplifiers (SOAs) was proposed by Vandoorne and colleague (Vandoorne, Dambre, et al. 2011; Vandoorne, Dierckx, et al. 2008) and subsequently built (Vandoorne, Mechet, et al. 2014). The photonic implementation of reservoir computing is based on the idea that photonic accelerators (Kitayama et al. 2019) can realize fast information processing with low learning costs (Antonik, Marsal, et al. 2019; Martinenghi et al. 2012; Paquot et al. 2012; Sugano, Kanno, and Uchida 2020). Some of the implementations use the scattering of light in complex media (Dong et al. 2020; Rafayelyan et al. 2020).

Spintronics RC. Some RC works used nanoscale electronics involving the charge and spin of electrons called *spintronics* (Wolf et al. 2001). Spintronics has been used in multiple ways to build RC models, using spin torque oscillators (STO) (Torrejon et al. 2017; Williame et al. 2019), or spin waves (Nakane, G. Tanaka, and Hirose 2018) for example.

2.3.1 Reservoir computing applications

RC has had successful applications in various fields. It has been applied for pattern classification; in audio with spoken digit recognition (D. Verstraeten et al. 2005) and waveform classification (Paquot et al. 2012); in computer vision with written digits recognition (Jalalvand, Van Wallendael, and Van De Walle 2015), and human motion classification (Antonik, Marsal, et al. 2019; Soh and Demiris 2012). Another popular application of RC is time-series forecasting (Jaeger 2001, 2002; Wyffels and B. Schrauwen 2010). RC has also been used in reinforcement learning to control agents in artificial environments (Kanno and Uchida 2022) or learn radio spectrum access (Chang et al. 2019).

2.3.2 Echo-state network

The echo-state network (ESN) (and simultaneously developed model *Liquid state machine* (LSM)) is the most well-known implementation of the RC principle (G. Tanaka

et al. 2019). The main idea behind this model is to drive a RNN with some input signal, which will excite the neurons within the reservoir and induce several nonlinear response signals. This is combined with a trainable linear regression to extract a desired output from a combination of these response signals. The ESN model is illustrated in Figure 2.17.

In practice, the input projection is often fixed, but it can also be trainable. The output can also be fed back to the reservoir as an input, and the structure of the RNN reservoir can be adapted to various applications.

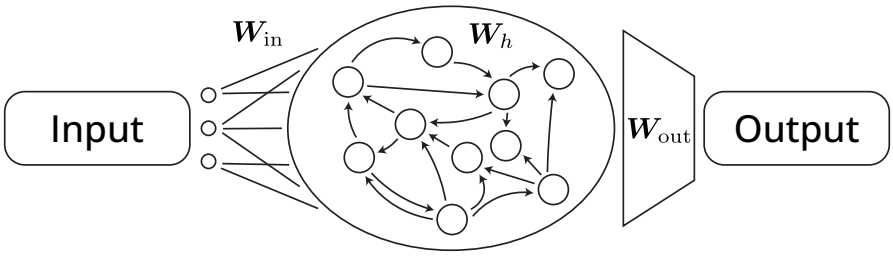


Figure 2.17 – Diagram of an echo-state network. The RNN is reprensented "flattened" in time. The circles represent the units of the hidden state of the RNN. An arrow between unit i and j corresponds to a non-zero entry in the matrix, $\boldsymbol{W}_{h,ij} \neq 0$. Non-linear operations are not represented.

We define an input sequence as $(\boldsymbol{x}_t)_{t=1}^{T} \in \mathbb{R}^{n \times T}$, where n is the input dimension and T is the number of input time steps. The initial state of the reservoir at $t = 0$ is $\boldsymbol{r}_0$. The echo-state network is based on the following update equation:

$$\boldsymbol{r}_{t+1} = (1 - \beta)\boldsymbol{r}_t + \beta \tanh(\boldsymbol{W}_h \, \boldsymbol{r}_t + \boldsymbol{W}_{\text{in}} \, \boldsymbol{x}_{t+1}), \tag{2.8}$$

where $\boldsymbol{r}_t$ is the K-dimensional state vector corresponding to hidden neurons — at time t, β is the leak rate, $\boldsymbol{W}_h \in \mathbb{R}^{K \times K}$ is a sparsely connected random hidden layer matrix and $\boldsymbol{W}_{\text{in}} \in \mathbb{R}^{L \times K}$ is the input projection matrix. The matrices $\boldsymbol{W}_h$ and $\boldsymbol{W}_{\text{in}}$ have random values.

For the random initialization of $\boldsymbol{W}_h$ and $\boldsymbol{W}_{\text{in}}$, Jaeger (2012) recommends: $\boldsymbol{W}_h$ should have an average of 10 non-zeros entries per row, all sampled uniformly in $[-1, 1]$. The matrix is then scaled to achieve a set spectral radius ρ, which is optimized for each experiment in (Jaeger 2012). $\boldsymbol{W}_{\text{in}}$ has its entries uniformly sampled in $[-1, 1]$. The columns of the matrix are then individually scaled with factors $\sigma_1, \ldots, \sigma_L$ specific to

each experiment. In this formulation, there are multiple parameters to optimize:

— The size of the reservoir K

— The spectral radius of $\boldsymbol{W}_h$, ρ

— The input weight scaling parameters $\sigma_1, \ldots, \sigma_L$

— The leaking rate β.

Jaeger (2012) explores three optimization schemes for these parameters:

Blind. The input weight scaling parameters are set to a single value σ. The parameters K, ρ, σ, and β are optimized. This corresponds to a search space of dimension 4.

Basic. Each σ_i is optimized individually or in groups. The other parameters are also optimized.

Smart. All parameters are optimized as in the **Basic** scheme, and the weights of $\boldsymbol{W}_h$ are also designed specifically for the target task.

The L-dimensional outputs are computed at times $t > 0$ as

$$\tilde{\boldsymbol{x}}_{t+1} = D(\boldsymbol{r}_t), \tag{2.9}$$

where $D : \mathbb{R}^K \to \mathbb{R}^L$ is a (trained) decoding function. For echo state networks, the decoder is often a linear transformation $D(\boldsymbol{r}_t) = \boldsymbol{W}_{\text{out}}\boldsymbol{r}_t$ where $\boldsymbol{W}_{\text{out}}$ is a matrix of dimensions $K \times L$. In our experiments in Chapter 6, we set $\beta = 0$, which was empirically observed to yield the best results on our tasks. Parameter β, as well as the randomly sampled weight matrix, are sometimes adjusted for each task (Jaeger 2012). We only use default values to obtain a task-independent setup with the least possible assumptions and to make the methods comparable.

2.3.3 Reservoir cellular automata

Cellular automata can also be used as a dynamical system in reservoir computing. Yilmaz originally proposed this in 2014 and later was named ReCA (Reservoir Cellular Automata) (Margem and Yilmaz 2017; Yilmaz 2014).

Several implementations of ReCA systems have been proposed and evaluated on various tasks (Babson and Teuscher 2019; Kleyko, Frady, and Sommer 2020; Margem and

Gedik 2018; McDonald 2017; Morán, Frasser, and Rosselló 2018; Nichele and Gundersen 2017; Nichele and Molund 2017a,b; Yilmaz 2014). In this section, we lay out a general description of reservoir computing with cellular automata based on these previous works and present our proposed extensions. We apply reservoir cellular automata to language-like tasks in Chapter 6.

Encoding. Input data is assumed to be categorical and sequential,

$$X = [X_1, \ldots, X_t, \ldots], \quad t \in \mathbb{N}, \quad \forall t \; X_t \in \mathcal{X} \subset \mathbb{N}. \tag{2.10}$$

A CA is a discrete system not specifically designed to handle input. The purpose of the encoding step in ReCA is to convert the input data points to vectors that can be embedded in the state space of the cellular automaton. Such encoding should translate — or embed — the input space $\mathcal{X}$ into a new one that can be combined with the current state of the cellular automaton. The goal is to make a cellular automaton "react" to its input and leverage the resulting computation to solve a problem.

Input projection. First, each categorical input vector is one-hot encoded into a vector of size L_{in}, where $L_{in} = |\mathcal{X}|$ is the number of input categories. We have

$$\forall t, \quad \boldsymbol{x}_t \in \{0,1\}^{L_{in}}, \tag{2.11}$$

with $\sum_{i=1}^{L_{in}} (\boldsymbol{x}_t)_i = 1$.

Next, the vectors $\boldsymbol{x}_t$ are projected onto the vectors $\boldsymbol{p}_t = P(\boldsymbol{x}_t) \in \mathcal{P} = \{0,1\}^{L_d}$ of fixed size L_d, where L_d is the size of the internal state of CA. Usually, we have $L_d > L_{in}$, which means that we embed the input in a space of higher dimension. In our work, we compute this projection in one of three ways:

One-to-one each input bit is assigned to a single index in $\boldsymbol{p}$. This is the projection function first proposed in Yilmaz (2014).

Building the reservoir The final state of the reservoir $\boldsymbol{r}_{t+1}$ is obtained by stacking I consecutive CA states obtained from a combination of inputs of a single state $\boldsymbol{s}'$ by applying Φ again. The parameter R can be understood as a space redundancy parameter,

while I is a time redundancy parameter. If the size of the state is n, the resulting reservoir vector $\boldsymbol{r}$ has dimension $K = I \times n$, where I is defined above. Similar to the ESN, the L-dimensional output tokens are calculated at times $t > 0$ as

$$\tilde{\boldsymbol{x}}_{t+1} = D(\boldsymbol{r}_t), \tag{2.12}$$

where $D : \mathbb{R}^K \to \mathbb{R}^L$ is the (trained) decoding function. Here, we predict $\tilde{\boldsymbol{x}}_{t+1}$, which corresponds to a language-modeling-like task of predicting the next input. It would also be possible to predict another sequence $(\boldsymbol{y}_t)_t$. Usually a linear decoding function is used such that $D(\boldsymbol{r}_t) = \boldsymbol{W}_{\text{out}}\boldsymbol{r}_t$. The complete pipeline of a RC model with CA is summarized in Figure 2.18.

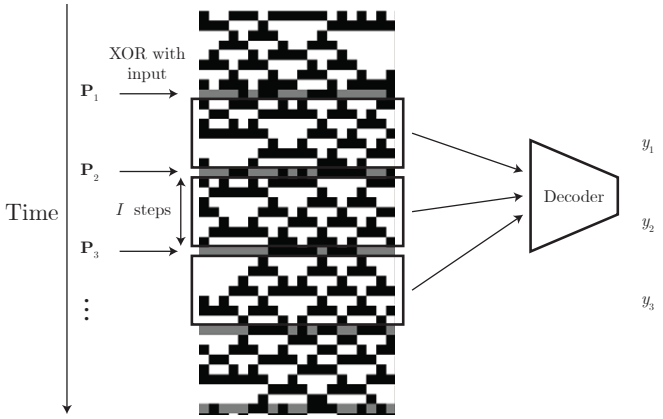


Figure 2.18 – Illustration of the 1D ReCA model with the XOR-based input and state combination method. The highlighted boxes correspond to I stacked CA states that make up a single reservoir state.

Chapter

3 Literature Review

In the previous chapter, we gave a detailed definition of cellular automata, neural networks, and reservoir computing, and described the interplay between them. The current chapter aims to provide an overview of several fields associated with learning and complex systems that have informed and influenced our work in this thesis. We begin by reviewing related work on complexity measures for dynamical and complex systems (Section 3.1). We also review works on defining and understanding emergence (Section 3.2), evolutionary methods for searching solutions in complex spaces (Section 3.3), methods for constructing open-ended evolving systems (Section 3.4), and methods for extracting and using the computations happening within complex systems (Section 3.5).

3.1 Measuring complexity

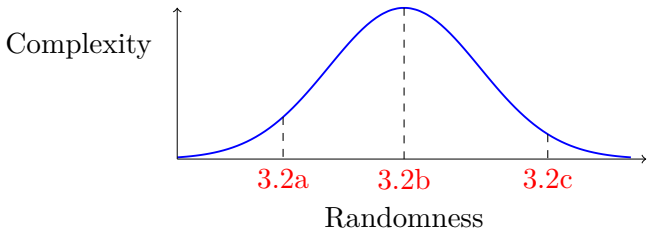


Figure 3.1 – *Ideal* "complexity" curve as a function of the "randomness" of a system. The three CAs from figure 3.2 are displayed at their approximate expected location on that curve. This curve is just for illustrative purposes and does not correspond to any true function linking randomness and complexity.

Measuring the complexity of a system is a fundamentally difficult task. Many complex systems exhibit what Peter Grassberger calls *self-generated complexity* (Grassberger 1986). This means that the formulation of the system is translationally invariant and the observed structure arises from a spontaneous breakdown of translational invariance. For

example, CA has a uniform update rule, but the complexity arises locally in some of them. Unfortunately, there is no universally accepted and formalized notion of "complexity", although most intuitively agree that it exists. For example, Figure 3.2 shows three examples of behaviors generated by 1D ECAs. Most people would consider the leftmost figure 3.2a to be not complex. However, depending on one's definition of complexity, the last picture could be labeled as complex.

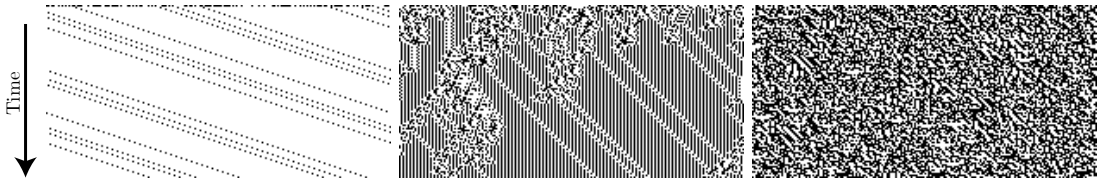


Figure 3.2 – Three 1D Cellular automata with qualitatively different behavior. The left (3.2a) and right (3.2c) CAs are usually not defined as "complex", whereas the middle CA (3.2b) appears to be more complex than the others. The Figures show CAs space-time diagrams. Each row represents the state of the CA at a single time step, with the time increasing from top to bottom.

No clear observable quantities and protocols have been proposed that would give a quantitative notion of complexity. Grassberger (1989a) argued that no single quantity is sufficient to measure complexity since it depends on exactly how the meaning is assigned to this term. Even when some meaning has been fixed by the definition of an observable quantity, the statistics of the measurement of this observable are also crucial to its interpretation (H. Gutowitz 1995).

Despite these drawbacks, we seek a measure of complexity that matches intuition as well as possible. Such a measure would assign low complexity to both simple objects and random objects and high complexity to objects in between. For illustration, we place the three ECAs from Figure 3.2 on this ideal complexity curve, shown in Figure 3.1. For example, in Chapter 4, we develop a complexity metric that matches the human intuition of complexity on a dataset of cellular automata.

One of the most widely known concepts of the complexity of symbol sequences, "algorithmic complexity," should also be called a measure of information. This is the position held by one of its co-creators (Gregory J. Chaitin 1990). In the literature, information measures are often used as complexity metrics. These quantities are better measures of "randomness" than complexity, but they are nonetheless important for the

study of complex systems. In this section, we list several complexity measures, including information measures that have influenced our research.

3.1.1 Information content

For an event E with probability P, the information content of this event is defined as the negative logarithm of its probability, that is,

$$I(E) := -\log(P). \tag{3.1}$$

This metric quantifies how unlikely an event is and depends on how the probability P was estimated in practice. This notion is at the foundation of information theory. The information content of a finite set of N events is maximal when all events are equally probable, that is $I(E) = \log(N)$.

3.1.2 Shannon Entropy

Shannon entropy is defined as the expected information content of the input (Shannon and Weaver 1975). We have

$$H(X) := \mathbb{E}[-\log(P(x))]_{x \in X}.$$

This measure is a lower bound on the number of bits that the input could be compressed to.

In the case of a 1D ECA, the Shannon entropy could be computed cell-wise over the time distribution of the states, producing an entropy per cell score that can be averaged over an entire automaton. This measure is, for example, one of the measures used by Wolfram in (Wolfram 1983) and by Langton in (Langton 1990) to study how the parameter λ affects the behavior of the automaton. There are several ways to compute it when dealing with a CA, depending on which part of the CA is considered as the main random variable. If the CA is finite, the state at timestep t can be seen as a random variable that can take one of 2^N possible values (with N the width of the automaton state). In that case, the probability of a state can easily be estimated by counting

its number of occurrences during evolution. For too large state spaces, this becomes challenging, as it is very unlikely that a given state will be seen again.

3.1.3 Rényi Entropy

The Rényi entropy is a generalization of Shannon entropy that gives different weights to events of various probabilities (Rényi 1961). It is formally defined for an $\alpha \geq 0, \alpha \neq 1$ and a random variable X with possible outcomes $0, 1, ..., n$ as

$$H_\alpha(X) = \frac{1}{1-\alpha} \log \left(\sum_{i=0}^{n} p_i^\alpha \right).$$

In the limit $\alpha \to 1$ it is equal to the Shannon entropy, where the probabilities of each event are equally important. $\alpha \to \infty$ yields the Min-entropy, and we have $H_\infty(X) = \min_i(\log(p_i))$. With $\alpha \to 0$ Rényi entropy is the same as Max-entropy, or topological entropy. Rényi entropy was used by Wolfram (1983) and Lindgren and G. Nordahl (1988) to estimate the complexity in infinite CAs.

3.1.4 Mutual information

The mutual information is a measure of the dependence between two variables. Two independent variables will have mutual information of zero. If the two are strongly dependent, for example, if one is a function of another, the mutual information between them will be larger.

Gregory J. Chaitin (1987) proposes to split a system into fixed blocks and calculate mutual information among its components. He uses the maximum value of the mutual information in all possible partitions to define "life" in a mathematical way. Shaw (1984) and Grassberger (1986) also use the mutual information measure between two semi-infinite blocks in a sequence to define complexity. Mutual information has also been used to study the complexity of CAs as the parameter λ is changed (for details on λ in the context of CA, see section 2.1.4) (H. A. Gutowitz and Langton 1988; Li, Norman H. Packard, and Langton 1990).

3.1.5 Computational complexity

Algorithms can be described by their space and time complexity, which respectively correspond to the amount of memory storage and CPU time necessary to run them (Hopcroft, Motwani, and Ullman 2007; Packel and Henryk Woźniakowski 1987; Traub, Wasilkowski, and H. Woźniakowski 1983). A problem that depend on a parameter N will be said to be NP-hard if the time needed to find a solution to them increases exponentially with the parameter N. NP-hard problems can be said to be complex as they require a supposedly irreducible amount of computation to compute their output. In practice, this description only applies to computations that were generated by a hand-designed algorithm and not the kind of computations self-generated by a complex system that we are interested in. For example, the computations generated by one CA cannot be said to be computationally more complex than those of another CA because the algorithm that implements them is fundamentally the same and computationally simple.

3.1.6 Solomonoff–Kolmogorov-Chaitin complexity (algorithmic complexity)

Introduced by Solomonoff (R. J. Solomonoff 1960), Kolmogorov (Kolmogorov 1968) and Chaitin (Gregory J Chaitin 1977, 1969, 1990), this complexity measure is defined for a string s of characters and a universal description language (e.g., a programming language) as the length of the shortest program that can generate the string s.

This number $K(s)$ is called the minimum description length of s. We note that Chaitin prefers to call his field "algorithmic information theory." His position is that "algorithmic complexity" is a measure of randomness rather than a measure of complexity (Gregory J. Chaitin 1990).

From the invariance theorem (Gregory J. Chaitin 1969; Kolmogorov 1968; R. Solomonoff 1964a,b), the difference in algorithmic complexity of the same string s in two different description languages is bounded, although this bound might be very large in practice.

The algorithmic complexity is uncomputable (Kolmogorov 1968; R. Solomonoff 1964a), and there exists strings of arbitrarily large complexity, which makes it hopeless to use this exact measure in practice. It can be approximated from above, but no accuracy

guarantee can be given, and the runtime of the approximators can grow arbitrarily large. Moreover, algorithmic complexity tells us how much information is required to encode a number, but does not tell us how difficult it is to recreate the number from that code (Gell-Mann 1988).

A range of resource-bounded approximations of algorithmic complexity were developed to obtain computable complexity measures (R. P. Daley 1973; R. Daley 1977; Feder, Merhav, and Gutman 1992; Ko 1986; Schmidhuber 2002). One rather straightforward way of approaching the algorithmic complexity of an arbitrary string is to use a compression algorithm and use the length of the decompression program plus the length of the compressed string as an upper bound to the algorithmic complexity. This is similar to the optimal symbol compression ratio (OSCR) algorithm proposed by (Evans, Saulnier, and Bush 2003). Zenil (2010) used the compression-based method to classify 1D ECA. This method, which often uses the popular LZ algorithm, can be seen as an independent complexity measure, also closely related to Lempel-Ziv complexity, described in more detail in the next section 3.1.7.

However, it is worth noting that when a complex system is described by an algorithm (such as for a CA), the algorithmic complexity is also easily upper bounded by a constant value entirely defined by the algorithm that simulated the complex system. For example, for a CA, the algorithmic complexity is lower than the size of the rule table of the automaton, its characteristics (size, boundary conditions, etc.), its initial state, and the number of simulation steps. This approximation does not inform us about the difference in complexity between the two CAs.

3.1.7 Lempel-Ziv complexity

The Lempel-Ziv complexity, as defined in (Lempel and Ziv 1976) is the number of steps in the LZ algorithm, directly related to the number of repeated substrings in the input string. The main idea of this algorithm is to scan the input string while trying to find some repetition of the previous input in the incoming data. This builds over time a set of basic components called the exhaustive history of the string, from which the complete string can be constructed. The number of components in that set is the Lempel-Ziv complexity of that string.

The compressed length method for measuring complexity uses a compression algorithm to reduce the size of the input. An input with regularities and repetitive patterns will produce a small output, whereas a completely random input will be irreducible to a simpler, shorter string.

3.1.8 Logical Depth

Bennett's logical depth is a measure of complexity based on the algorithmic complexity (C. H. Bennett 1988, 1995). The main difference is that it takes into account the computation time (or number of steps) along with the length of the program used to generate the sequence. It is a combination of algorithmic complexity and computational complexity. For a universal computer U, the logical depth of a string x at a significance level s is defined as

$$\min\{T(p) : (|p| - |p^*| < s) \wedge (U(p) = x)\}, \tag{3.2}$$

where T is the number of steps that the universal computer runs the program p for. This quantity corresponds to the least time required to compute the string x by a s-incompressible program, that is a program p whose length is within s symbols from the optimal program p^* of the algorithmic complexity metric.

Luis Antunes et al. (2006) considered logical depth to be one instance of a more general concept: *computational depth*; and proposed several other variants.

3.1.9 Thermodynamic Depth

The thermodynamic depth, developed by Lloyd and Pagels (1988) is another measure of complexity based on the intuitive notion that complex systems lie somewhere in the continuum between order and chaos (Ceccatto and Huberman 1988; Gregory J. Chaitin 1990; Deutsch 1985). Similar to computational complexity and logical depth, this metric is designed to be a measure of how the system came to be in its final state. The complexity corresponds to how difficult it is to put the system together in that state.

By the definition of thermodynamic depth, the average complexity of a state must be proportional to the Shannon entropy (Shannon and Weaver 1975) of the set of trajectories

that the experiment determines can lead to that state. If we write p_i the probability of reaching the target state with the i-th possible trajectory, we have

$$S = -\left(\sum_i p_i \log p_i\right). \tag{3.3}$$

The measure of complexity of a macroscopic state s of a system that has reached that state by the i-th possible trajectory is $-k(\log p_i)$, where p_i is the probability that the system has reached s by the i-th trajectory, and k is an arbitrary positive constant. The thermodynamic depth of the state s is defined as

$$\mathcal{D}(s) = -k(\log p_i). \tag{3.4}$$

The quantity $\mathcal{D}$ can be thought of as the amount of information required to specify the trajectory that the system has followed to its present state. The thermodynamic depth of the whole system is then

$$\mathcal{D} = \sum_s \mathbb{P}(s) D(s), \tag{3.5}$$

where $\mathbb{P}(s)$ is the probability that the system followed the trajectory s.

James P. Crutchfield and C. R. Shalizi (1999) argued that the thermodynamic depth is a fundamentally flawed structural complexity measure because it relies on a set of chosen macroscopic states for the system, which are difficult to choose and need to be defined separately for each system. Lloyd and Pagels (1988) does not mention how these states are supposed to be chosen, nor do any follow-up work, making this complexity metric difficult to use in practice.

3.1.10 Epsilon-machines

The ϵ-machine is an approach to complexity that seeks to construct a metric more suitable for physical systems that also addresses some issues of other existing complexity metrics (James P. Crutchfield 2012). It is the result of the field of computational mechanics, an extension of statistical mechanics that describes not only the statistical properties of a system but also how it stores information and how it computes (James P. Crutchfield 1994, 2012; James P. Crutchfield and Young 1989; Feldman and James P

Crutchfield 1998). Computational mechanics algorithms take as input the time series being analyzed and output a minimal, optimized model that can reproduce a time series that is statistically equivalent to the input time series. The size of the model produces a metric known as statistical complexity. The models are known as ϵ-machines. Three optimality theorems say that ϵ-machines capture all the properties of a process (James P. Crutchfield and C. R. Shalizi 1999; James P. Crutchfield and Young 1989; C. R. Shalizi and James P. Crutchfield 2001): prediction: the ϵ-machine is its optimal predictor; minimality: compared to all other optimal predictors, the ϵ-machine of a process is its minimal representation; uniqueness: any minimal optimal predictor is equivalent to the ϵ-machine.

This model has many attractive properties and was successfully applied to study the symbolic dynamics of chaotic systems (James P. Crutchfield 1994), molecular dynamics (Ryabov and Nerukh 2011), single-molecule microscopy (Kelly et al. 2012), and the spatiotemporal complexity of CAs (James P. Crutchfield and Hanson 1993; Hanson and James P. Crutchfield 1997; C. R. Shalizi, K. L. Shalizi, and Haslinger 2004). The main drawback is the construction of the ϵ-machine itself, for which there is no general-purpose approach.

3.1.11 Sophistication

Intuitively, *sophistication* is the complexity in a set of strings of which the string is a "typical" member. Mota et al. (2013) defines sophistication based on the original definition of Koppel (1988) and Koppel and Atlan (1991). It measures the amount of structural information contained in a string. It also uses a follow-up result by (Vitanyi 2006) that shows that Koppel's definition is equivalent to measuring the complexity of a good model for a string up to low order terms. The definition uses an intermediate quantity called *discrepancy* that measures how far a set S is from being a good model of a string x.

Formally, the discrepancy Δ is defined for a string x and a set S that contains x as

$$\Delta(x|S) := \log |S| - K(x) + K(S), \tag{3.6}$$

where K is the Solomonoff–Kolmogorov-Chaitin complexity function.

The sophistication of x is defined as the complexity of the simplest model of x with a limited discrepancy:

$$\text{soph}_c(x) := \min_S \{K(S) : \Delta(x|S) \leq c\} \tag{3.7}$$

where the significance level c tells us how much discrepancy the set S is allowed to have.

Koppel's definition of sophistication, Definition 3.1, may not be stable. Small changes in c could cause large changes in $\text{soph}_c(x)$. For this reason, Luís Antunes and Fortnow (2009) introduces a new notion of *coarse sophistication* that incorporates the "constant" c as a penalty in the formula to obtain a more robust measure. In the formalism of Mota et al. (2013) the definition is

$$\text{csoph}(x) = \min_S \{K(S) + \Delta(x|S)\} \tag{3.8}$$

which is also equivalent to

$$\text{csoph}(x) = \min_c \{\text{soph}_c(x) + c\}\,. \tag{3.9}$$

Like algorithmic complexity, sophistication is a useful theoretical notion to model ideas of entropy and complexity, but it cannot be directly applied in numerical simulations and can only be approximated because it is also uncomputable.

3.2 Emergence

The notion of emergence is central to the study of complex systems. It is a broad concept that could be phrased as *the properties a composite entity acquires that its composing parts did not possess.* Emergence is often described as being similar to self-organization. Although these two concepts are closely related, they are fundamentally different ideas (De Wolf and Holvoet 2005). Both properties can exist independently within a dynamical system. Some examples of emergent behavior include:

— The spontaneous formation of clusters of randomly distributed objects - a behavior common in ant colonies forming bridges out of individual ants or birds flocking into

"murmurations" - that naturally emerges out of a simple set of autonomous actions having nothing to do with that clustering. This was demonstrated by Beckers, O. E. Holland, and Deneubourg (2000) in the context of exploring collective robotics). The macroscopic behavior in each of these examples is unexpected even though the details of the microscopic dynamics are well-defined.

— The spirals of the Belousov- Zhabotinsky chemical reaction (Tyson 2013).

— The Navier-Stokes-like macroscopic behavior of a lattice gas that consists, of simple unit-bit billiards moving back and forth between discrete nodes along discrete links at the microscopic scale (Hasslacher 1987).

There seems to be qualitatively different types of emergence, which de Haan (2006) describes as Discovery, Mechanistic emergence, and Reflective emergence. These correspond to different levels, or "strengths", of emergence, from fractal patterns to complex social systems and ecosystems.

A subset of emergence research has focused on complex adaptive systems. In such systems, emergence is explicitly used to refer to macro-level patterns arising from interacting agents (J. H. Holland 2000; S. Kauffman 1995; Langton 1986).

3.3 Evolutionary algorithms

Evolutionary algorithms are the class of algorithms inspired by natural biological processes, in particular evolution. There are many types of evolutionary algorithms, from genetic algorithms to quality diversity and novelty search (Lehman and Stanley 2011a,b). Evolutionary algorithms are used as search methods, in particular when the search space is hard to parametrize or very large (Poli and Logan 1996). Evolutionary algorithms facilitate the search for complex optimization spaces by using intermediate representations, as illustrated in Figure 3.3. CA rules are a good example of such search spaces. The CA rule space is very large, and it is difficult to predict how perturbations of the rule translate to changes in the behavior of a CA. Genetic algorithms were successfully used to evolve rules to perform complex computations (Melanie Mitchell, James P Crutchfield, and Das 1996).

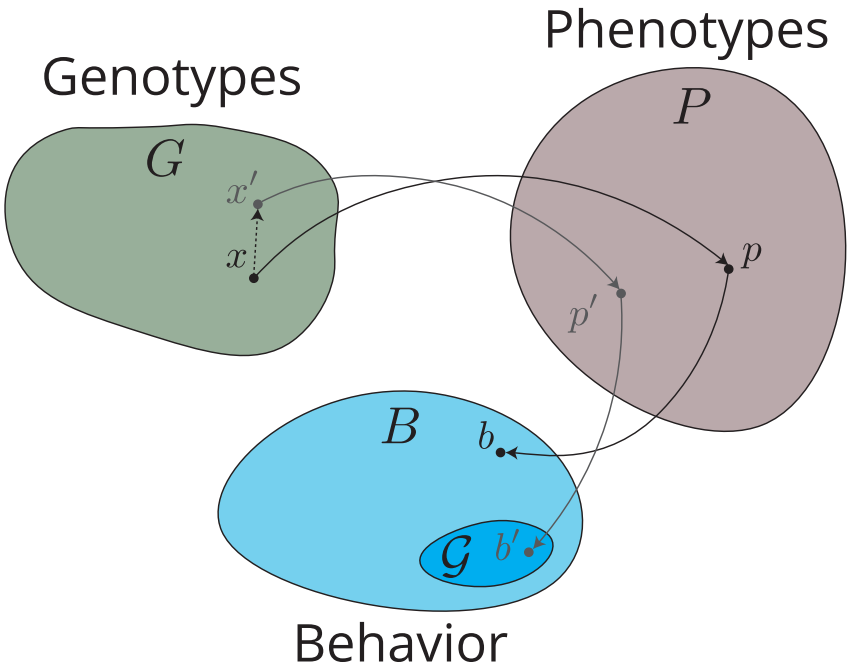


Figure 3.3 – Illustration of the general principle common to several evolutionary algorithms. There are three virtual spaces: (G) the genotype space where the candidate solutions are generated through some encoding. This is the space where the search process is happening. (P) the phenotype space, which corresponds to the decoded solution from a genotype x. It can be the policy of an agent, the rule of a CA, etc. (B), the behavior space, which is the space from which candidate solutions are evaluated. The smaller subspace $\mathcal{G}$ is the goal space, corresponding to the set of behaviors that would be considered successful. A small perturbation in the genotype will have potentially large consequences on the behavior of the candidate solution. In the case of a CA rule, the genotype could be a vector (x) that encodes a rule (p) that will translate into the CA behavior b.

3.3.1 Genetic programming

Many of the problems that machine learning and artificial intelligence are attempting to solve require the discovery of a computer program that produces some desired mapping between inputs and outputs. Solving the problems therefore amounts to searching the space of computer programs in order to find a suitable individual with high enough fitness (Banzhaf et al. 1998; Booker, Goldberg, and J. H. Holland 1989; J. R. Koza 1994; Langdon and Poli 2002).

In genetic programming, candidate solutions to a problem are encoded as *chromosomes*, which is a structured data representation that can be modified incrementally. In practice, it is often chosen to be a string of bits. Every generation, a population of solutions is evaluated. The best solutions are kept and combined to form the candidates for the next generation. Random mutations may also be applied to introduce randomness in the search.

3.3.2 Novelty search

The idea behind novelty search (NS) is to drive a search algorithm only by the novelty of produced behavior (Lehman and Stanley 2011a). The most counter-intuitive feature of this algorithm is that the actual objective of the task is not taken into account at all during the search process. Generated agents are evaluated solely on the novelty of their behavior. Despite this, it appears to be at least as efficient as other search processes that are focused on goals, such as maze navigation and biped locomotion (Lehman and Stanley 2011a), swarm robotics (Gomes, Urbano, and Christensen 2013), or neural network design (Risi, Hughes, and Stanley 2010). NS can lead to the discovery of innovative and creative solutions to complex and deceptive objectives that would be difficult or impossible to achieve using traditional optimization methods. We illustrate this property in Figure 3.4.

The principle of guiding search by novelty alone is closely related to the goal of this thesis, and especially with the challenge of parametrizing and searching the space of available complex systems without using any explicit goal function. The purpose of NS is to perform a search without a goal function, and we draw inspiration from this algorithm throughout our work.

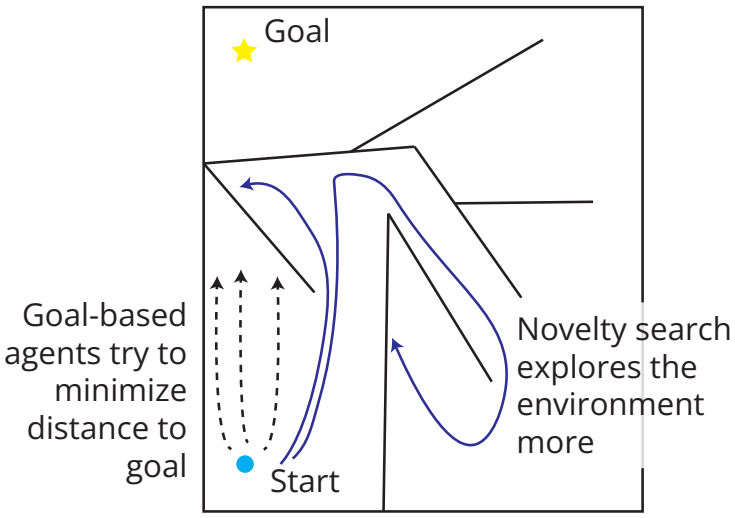


Figure 3.4 – Illustration of the advantage of novelty search (NS) on a maze environment with deceptive dead-ends. Dotted trajectories illustrate the behavior of a goal-based search, which will start by minimizing the distance to the goal. Blue trajectories are from a novelty search algorithm exploring the space more efficiently. This type of maze environment was proposed by Lehman and Stanley (2011a).

This method has some limits, which are caused by a relative lack of theoretical understanding of the effects of the parameters of NS. The choice of behavior descriptors for determining the novelty of agents is also not straightforward. Several empirical studies have investigated the impact of a range of parameters on the performance of NS (Gomes, Mariano, and Christensen 2015; Kistemaker and Whiteson 2011), while other works have begun to shed light on the theory (Doncieux, Laflaquière, and Coninx 2019). In practice NS algorithms are expected to cover the exploration space well (Cully and Demiris 2017; Pugh, Soros, and Stanley 2016), in some cases uniformly (Gomes, Mariano, and Christensen 2015). NS algorithms are related to genetic algorithms since they maintain a population, that is, a set of individuals used to measure the novelty of current solutions. There is also an archive of the behavior of previous individuals to ensure that the novelty is measured against all previously generated behaviors. There exists multiple strategies for maintaining this archive (Gomes, Mariano, and Christensen 2015). Some use individuals whose novelty was above a threshold when first evaluated (Lehman and Stanley 2011a), the most novel individuals of each generation (Liapis, Yannakakis, and Togelius 2015), random individuals from the past generations (Lehman and Stanley 2010), or none at all (Mouret and Doncieux 2012). An alternative to

explicitly measuring novelty in NS is *fitness sharing*. The goal of fitness sharing is to encourage the diversity of a population by making them share virtual "resources" (Goldberg 1987; J. H. Holland 1992). A particular implementation of NS using this idea, with rewards physically spread across the environment, was done by Herel et al. (2022).

Typically, novelty search algorithms use one of a few types of maps from genotypes to phenotypes, that is, a map from the space of encoded solution to agent and policy implementation (Mouret and Doncieux 2012). Some common ones are:

— A RNN (of type Elman 1990 or Jordan 1997), for which the weights are directly searched. For input size n_i output size n_o, and hidden size n_h, $n_i n_h + n_h n_o + n_h^2$ weights must be evolved.

— Neuro-evolution of augmenting topology (NEAT, Stanley and Miikkulainen 2002), which is an encoding of both neural network weights and architectures that has a built-in mechanism to sustain diversity.

3.4 Open-ended evolution

The field of Artificial Life research has worked to figure out what the fundamental conditions for the emergence of living systems are, and how to create a process that can display analogous levels of creativity and complexity as natural evolution (Dyson 1999; Eigen and Schuster 1979; Langton 1989; N. Packard et al. 2019; Soros, Stanley, and Lehman 2017; Stanley 2019). It builds upon the data and understanding we collected about the process of life, but abstracts from any specific living process and attempts to integrate various approaches into one unified research to extract the first principles of life. A major assumption underpinning this research is that this natural evolutionary process can be implemented equally well in different media (Dennett 1996).

A system that behaves like natural evolution, producing a seemingly endless amount of novelty and complexity starting from elementary building blocks is called *open-ended*. The main challenge of open-ended evolution (OEE) research is that there is no single simple test for the phenomenon, but instead different kinds of open-ended evolution exist. Systems can exhibit more than one kind at a time. In the report from a workshop on OEE at York (T. Taylor et al. 2016), the authors summarized the different types of OEE, which were further refined in a follow-up work (N. Packard et al. 2019). They are:

1. Interesting new kinds of entities and interactions
2. Evolution of evolvability
3. Major transitions
4. Semantic evolution.

The first category describes the ability of a system to construct new entities with different properties, behaviors, or interactions with other entities. For example, the Tierra simulation sees entities emerge that exploit the computing power of others, acting as parasites (Ray 1991). The second type is related to how open-ended evolution itself can be evolved through the emergence of multiple "stepping stones" that allow individuals to use higher-level evolutionary units or through interactions (Pattee and Sayama 2019). "Major transitions" refer to the emergence of new levels of hierarchy in evolving populations. These transitions are characterized by groups of reproducing entities that interact tightly and form a new population of higher-level reproducing entities. The process can repeat, with certain groups in the new population forming even higher-level entities. This process of emergence of major transitions was explored in (Moreno and Ofria 2019; Sayama 2019) for example. Lastly, semantic evolution is related to the work of Ikegami, Hashimoto, and Oka (2019), who propose a new category of open-ended evolution called semantic evolution, which refers to the evolution of semantic relationships within a system. This can be observed in the evolution of web services, where the meaning of tags changes as new combinations of tags are created. It is not biological in nature and is also present in the analysis of technological evolution (Bedau, Gigliotti, et al. 2019), where the meaning and importance of keywords shift over time.

3.4.1 Defining open-endedness

Several necessary conditions and requirements have been identified in the OEE literature, forming an overlapping set of potential research directions for developing open-ended systems. We list a few of these conditions here. First, Maley (1999) identifies four requirements:

1. An open-ended evolutionary system must demonstrate unbounded diversity during its growth phase.

2. An open-ended evolutionary system must embody selection.
3. An open-ended evolutionary system must exhibit continuing new adaptive activity.
4. An open-ended evolutionary system must have an endogenous implementation of niches.

Next, Soros and Stanley (2014) identified four more necessary conditions, which are

1. A rule should be enforced that individuals must meet some minimal criterion (MC) before they can reproduce, and that criterion must be nontrivial.
2. The evolution of new individuals should create novel opportunities for satisfying the MC.
3. Decisions about how and where individuals interact with the world should be made by the individuals themselves.
4. The potential size and complexity of the individuals' phenotypes should be (in principle) unbounded.

Then, T. Taylor (2015) also stated five requirements for an open-ended system:

1. Robustly reproductive individuals.
2. A medium allowing the possible existence of a practically unlimited diversity of individuals and interactions, at various levels of complexity.
3. Individuals capable of producing more complex offspring.
4. An evolutionary search space that typically offers mutational pathways from one viable individual to another viable (and potentially fitter) individuals.
5. Drive for continued evolution.

Taylor also states a more general condition that should be sufficient for creating an open-ended system as "evolutionary dynamics in which new, surprising, and sometimes more complex organisms continue to appear" (T. Taylor 2015; T. Taylor et al. 2016).

All these conditions illustrate a major challenge of OEE research: the lack of a clear definition or notion of what exact conditions make a system open-ended. We seem to agree about what is not open-ended, but whenever a constraint or requirement for OEE is identified, subsequent evidence forces us to refine them later. This is related to the problem of complexity, for which no single definition exists (Johnson 2009).

A common process to produce systems that behave analogously to natural evolution is to start from evolvable units or building blocks (Channon 2003; Ofria and Wilke 2004; Ray 1991; Sims 1994; Soros and Stanley 2014; Spector, Klein, and Feinstein 2007; Yaeger 1994). The reason is that starting from higher-level primitive units whose emergence would be hard to characterize may be easier and faster than starting from lower-level components. The results obtained from these systems are often surprising, as they bear some key similarities to natural evolutionary processes. For example, we note the emergence of parasitic entities within the Tierra simulation (Ray 1991). The process of emergence of these building blocks from simple rules and components has also been investigated significantly (Bagley and Farmer 1991; Flamm et al. 2010; Hutton 2007; Sayama 2011). It appears harder to bridge the gap and create high-level evolutionary-like processes and behaviors from elementary rules and substrates.

3.4.2 Open-endedness in cellular automata

One class of systems that has rich interactions between each of its components, as well as no predefined evolvable units or assumptions about individuality is the CA. One of the very first automata, Von Neumann's self-reproducing machine, was designed with goals that align with open-ended evolution, which is to build a machine with no central controller and limited local interaction that can self-reproduce as a whole (Pesavento 1995; Von Neumann and Burks 1966). Later, other self-replicating structures such as Langton's loop (Langton 1984) and evoloops (Salzberg and Sayama 2004; Sayama 1999) showed that more properties of open-ended systems can be included in a CA. A potential limitation of CAs is the absence of the notion of conservation of "matter". For example, the game of life can start from a configuration with very few live cells and create many more at no cost during its evolution. Some authors believe that this conservation property is essential to the construction of an open-ended evolving system (T. Taylor 2002).

3.4.3 Artificial chemistries

Artificial chemistries, are computational models that aim to simulate the behavior of chemical systems, which are typically used to study the emergence of complex behav-

ior from simple interactions between individual molecules. Most of the time, artificial chemistries (AC) do not try to accurately model chemical processes (as in (Bedau, McCaskill, et al. 2000; Buliga and L. H. Kauffman 2014; Ostrovsky et al. 2001; Sayama 2011) for example). Instead, they build models of the dynamics of complex molecular processes that lead to evolutionary behavior (Dittrich, Ziegler, and Banzhaf 2001). By abstracting away from natural molecular processes, AC tries to uncover fundamental conditions for the emergence of organization, self-maintenance, or self-construction with basic building blocks. There are various approaches for building these artificial chemistries, of which we review a few here.

Rewriting systems. Rewriting systems are a class of computational models that use a set of rewriting rules to transform and manipulate strings of symbols. They are composed of entities or symbols that get modified according to their set of syntactic rules. Patterns of symbols or entities are replaced according to these rules. In AlChemy (Fontana and Buss 1994), molecules are represented as expressions in the λ-calculus. The λ-calculus is a mathematical formalism that, like Turing machines, is capable of describing any computable function. In AlChemy, pairs of randomly selected expressions are joined using function application, evaluated, and the resulting value is added back to the population of molecules. This process is repeated to simulate the behavior of a chemical system. Kruszewski and Mikolov (2022) hypothesized that self-reproducing metabolisms emerge as a subset of autocatalyzed reactions within a Turing-complete set. They introduced *combinatory chemistry*, an artificial chemistry designed to capture the core computational properties of living systems that is based on the rewriting rules of *combinatory logic* (Curry 1958; Schönfinkel 1924). The generated structures exhibit behaviors that are similar to natural metabolisms. The system is able to represent patterns of arbitrary complexity, and it is able to produce diverse self-reproducing patterns.

Cellular automata. Cellular automata (CAs) can be seen as a particular case of lattice molecular systems. For example, the autopoietic system (referring to the ability of an organism to continuously sustain itself through the exchange of matter and energy with its environment, while maintaining its own boundaries and identity) introduced by

F. G. Varela, Maturana, and Uribe (1991) is a 2D square grid with sites that be occupied by *atoms* which is similar to a CA with 4 states. Each of these states is analogous to a chemical component of the system: ($\emptyset$) empty site, (S) substrate, (K) catalyst, and (L) monomer. Basic rules are applied asynchronously and define how neighboring atoms interact with each other. Remarkably, stable self-repairing cells spontaneously arise from these basic rules. Their membrane is composed of a chain of monomers, which is maintained by the substrate and catalyst reacting according to the rules. Some key components of that model were investigated in other works, showing that they are crucial for this emergence autopoeisis to be possible (McMullin and F. J. Varela 1997; Zeleny 1977).

3.5 Computing with complex systems

The problem of computing within complex systems is closely related to the question of decentralized parallel computation, in general, for which there exists abundant literature. Different paradigms exist for controlling and harvesting the computations within complex systems. Several other names have been used for closely related topics, such as organic computing (Müller-Schloer 2011) which is the study of systems with life-like properties such as self-organization or the ability to adapt to a dynamically changing environment. Agent-based computing (Jennings 1999) focuses on computing systems composed of several relatively simple autonomous agents. Amorphous computing (Abelson et al. 2000; Nagpal 2002, 2008) is about making large amounts of individual computing elements work and ensuring "the cooperation of large numbers of unreliable parts interconnected in unknown, irregular, and time-varying ways".

3.5.1 Computing in cellular automata

Cellular automata (CAs) are decentralized parallel systems with many identical components with local connectivity. Because of these properties, they have the potential to carry out robust and efficient computations. CA-based computing machines could recover from perturbations or carry out computations in stochastic environments. Moreover, they are also interesting for modeling the behavior of natural complex systems. For more details on CAs, see Section 2.1.

Von Neumann's self-reproducing CA. The early developments of CAs were guided by Von Neumann's question "What kind of logical organization is sufficient for an automaton to be able to reproduce itself?" Since this question may admit very simple solutions, additional constraints were added so that the problem does not admit trivial solutions. For example, Von Neumann required that the automaton in question be equivalent in power to a universal Turing machine while having minimal complexity (Von Neumann and Burks 1966). The final model he proposed was an intricate CA composed of a tape containing the instructions to construct the next automaton and a construction unit that would progressively build that new automaton cell by cell. This was an early example of complex computations being carried out by a relatively simple decentralized system such as a CA.

Universal Computation in Cellular Automata. It is not difficult to see that a CA can be capable of universal computation. The basic approach is to show that it can simulate a Turing machine, which we assume has an infinite tape. A CA rule can be constructed by reproducing all the steps of a Turing machine's behavior while adding a state to each cell, indicating whether it is active or not, and enforcing that only one cell is active at a time. This turns the parallel CA into a sequential object that simulates a Turing machine. A similar construction was carried out by (A. R. Smith 1971), making a CA with one-to-one correspondence between its time steps and that of a target Turing machine.

Another approach was used to show the universality of the game of life (see details in Section 2.1.3). Instead of simulating a Turing machine, basic logical functions are built from gliders (see figure 2.5a) produced by a game of life structure called *glider gun*. Two gliders that collide within the simulation will either annihilate or keep moving depending on the exact position in which they collide. This allows the construction of NOT, AND and OR gates. For example, see Figure 3.5 for a construction of a NOT gate in the game of life.

Universal computation is an intriguing yet impractical property, and proving that a system is a universal computers often has limited usability, but it serves as a proof of principle that the studied system is theoretically as powerful as any computer. In

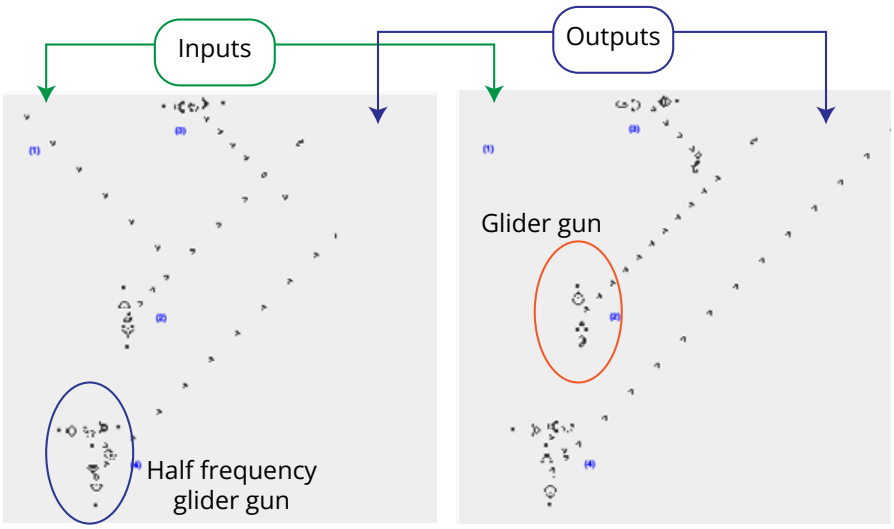


Figure 3.5 – Construction of a NOT gate in game of life. The left image represents an input equal to 1, while the second shows an input equal to 0. Output streams behave like a NOT gate. A glider gun and a modified half-frequency glider gun are highlighted. They are necessary for the construction of the gate. Images are adapted from a blog post (Carlini 2020).

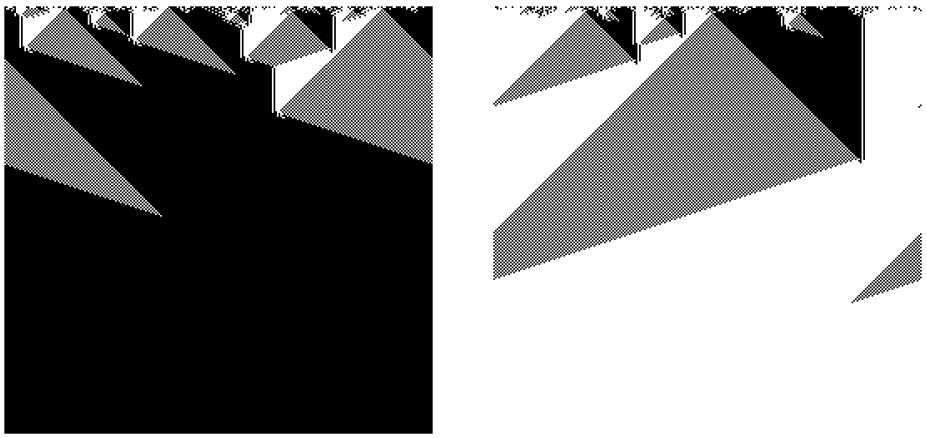
the case of the game of life, this was illustrated by the implementation of the OTCA metapixel which is a tileable square cell object that can be used to simulate another instance of game of life within the game. It is important to note that such embedded computers and Turing machines are often very slow and inefficient, and the game of life has brittle structures which would be destroyed by the slightest perturbation, making these constructions unusable for anything else than demonstration.

Evolving cellular automata with genetic algorithms. An important advancement beyond the clever yet challenging manual design of computational rules in CAs is the utilization of learning algorithms for automated rule design (Melanie Mitchell, James P Crutchfield, and Das 1996). Genetic algorithms, which are search methods inspired by biological evolution, are one such type of learning algorithm (Booker, Goldberg, and J. H. Holland 1989). See Section 3.3.1 for more details about genetic algorithms.

Some early work on evolving CAs was done by Norman Packard and colleagues (Norman H. Packard 1988; Richards, Meyer, and Norman H. Packard 1990). Richards, Meyer, and Norman H. Packard (1990) used genetic algorithms to learn the rules of CA to match experimental data for the patterns created by dendritic solidification of NH_4Br. In (Norman H. Packard 1988), the author evolves CAs to perform a specific computation,

and observes that the evolution process tends to select CAs with a λ parameter close to the critical value observed by Langton that corresponds to the edge of chaos (see Section 2.1.4 for details about the parameter). However, other authors were unable to replicate this experiment later (Melanie Mitchell, P. Hraber, and James P. Crutchfield 1993). J. Koza (1992) applied genetic algorithms to CAs to generate random numbers.

The problem of density classification was thoroughly explored since it fits the CA paradigm particularly well (Andre, F. H. Bennett, and J. R. Koza 1996; J. P. Crutchfield and M. Mitchell 1995; Das, Melanie Mitchell, and James P. Crutchfield 1994; Melanie Mitchell, James P. Crutchfield, and P. T. Hraber 1994; Melanie Mitchell, P. Hraber, and James P. Crutchfield 1993; Sipper 1996). The goal of this task is to have CAs figure out what the most abundant cell state in an initial binary configuration in one dimension is. This task is trivial for a central controller that can read the state of all cells in the initial grid, but it is much more complex for decentralized systems like CAs that have a small radius of interaction with neighbors. The evolved CAs developed various strategies to solve the task. Some are unsophisticated, such as expanding the 1s or 0s, which does not work on all validation examples. We show one of the more sophisticated solutions discovered during some runs of the genetic algorithm in figure 3.6. That solution uses the propagation of structures within the CA state space to communicate the local state to the rest of the grid, which usually ends with the right answer from the CA.



(a) Density > 0.5 **(b)** Density < 0.5

Figure 3.6 – The particule-based solution to the density classification problem (Melanie Mitchell, James P Crutchfield, and Das 1996). When the density is above 0.5, the CA quickly reaches a state with only black cells, whereas when the density is lower, it goes to a state with white cells only.

Reliable computation in cellular automata. The problem of carrying out reliable computations in CAs is studied early on in its history because of the potential of that model for a real hardware implementation. The first studies used probabilistic approaches for solving the error detection and correction problems of automata — which includes other models than CAs (Harao and Noguchi 1973; Neumann 1956; Tsertsvadze 1964; Winograd and Cowan 1963). Other methods make use of the special structure of CAs, but rely on strong assumptions about the likelihood of errors (Harao and Noguchi 1975; Nishio and Kobuchi 1975). Peter Gács has shown how to construct a 1D CA which can perform arbitrarily large reliable computations, assuming that each cell gets an error with a positive probability (Gács 1986). The fault model he describes is important from the point of view of ergodic theory because Gács' result invalidates the "positive probability conjecture" in statistical physics, which states that all one dimensional infinite particle systems with positive transition probabilities are ergodic, which means that it will visit all parts of its state space eventually. For more recent work on fault tolerance by Gács, see (Gács 2001).

Neural cellular automata. The principle of NCA is based on the analogy between CAs and two types of neural networks: RNNs and CNNs (see Section 2.2 for more details about the analogy). Following this analogy, CAs can be formulated as a continuous recurrent and convolutional neural network system called a neural cellular automaton (NCA). NCAs can be manipulated like neural networks, using automatic differentiation, backpropagation, and optimization algorithms to make these models learn some target task. The first example used this neural network representation to learn a NCA version of existing CA rules from recorded examples of their evolution (Gilpin 2018). The structure of the training process and the final error of the resulting model are used by the author as a tool to understand the complexity and properties of the original CA.

NCAs were also used to learn to produce stable self-repairing patterns that resemble a target image (Mordvintsev et al. 2020). The training process is structured in such a way that the NCA needs to learn to maintain the pattern stable across time and space, but also recover from random perturbations that are introduced at arbitrary points in time. This results in an interesting demo where a pattern is generated and maintained

in real-time by a NCA while a user can directly apply perturbations *.

(a) Perturbed patterns

(b) Recovered patterns after a few hundred NCA steps.

Figure 3.7 – Resistance of patterns to perturbations with NCAs. NCA can robustly maintain patterns, even under relatively strong perturbations (10-20% of the pattern's size). This image was generated with an online notebook based on code from Alexander Mordvintsev †.

The same authors also used NCAs to learn to classify digits from the MNIST database of handwritten digits (Randazzo et al. 2020). The particularity of this model is its ability to perform the classification in a decentralized way, by having the NCA change the color of the digits directly on the CA grid. Cells can only communicate with their direct neighbors, so the model has to reach a consensus by spreading information across the grid. This is done in parallel, so multiple digits can be classified simultaneously through this progressive consensus process ‡. Another work used the same model to learn to continuously generate images with the same texture as a target (Niklasson et al. 2021). Again, an advantage of this method is the ability to recover from perturbations.

Figure 3.8 – Examples of stable hybrids obtained with NCAs trained with the method of (Cisneros 2021).

We used NCAs as part of our submission to the Minecraft open-endedness challenge

*. https://distill.pub/2020/growing-ca/

†. https://colab.research.google.com/github/google-research/self-organising-systems/blob/master/notebooks/growing_ca.ipynb

‡. https://distill.pub/2020/selforg/mnist/

2021 §, a competition organized at the Gecco 2021 conference competition track ¶ (Cisneros 2021). The goal was to build an open-ended system within the computer game Minecraft. This means building a systems that fulfill the following requirements:

— Divergence: Open-ended algorithms are not expected to slow down or converge but rather keep expanding and generating more complex outputs over time.

— Diversity: Does the algorithm produce entities with strong phenotypic diversity?

— Complexity: Can the algorithm produce complex entities or entities interactions that give rise to complex systems? Are hierarchical and modular structures present?

— Ecological interactions: Do the created entities interact with each other?

— Life-Like properties: Inspiration may be taken from other attributes of living systems.

In each of the examples in Figure 3.7, a separate CA was trained on a pattern starting from a single black pixel. We extend this setup by training a single NCA on multiple patterns from different seeds. Instead of a single black cell, a small number of black cells are arranged in simple shapes. We call these patterns *seeds*. NCAs are neural networks with many redundant parameters. This allowed us to learn (and encode) more than one pattern (or larger patterns) without adding parameters in the architecture. The resulting patterns are slightly lower quality but still stable under perturbations and capable of growing from their corresponding seed pattern.

We turned the process of combining seed patterns into a "game" by allowing user interaction with the evolving CA through the Minecraft video game, using an API to communicate between the program and the game. Our system is not open-ended by itself, but serves as an exploration space and uses human interactions to play with the shapes and seed patterns. This is in the spirit of open-ended game-like systems like Picbreeder (Secretan et al. 2011; Woolley and Stanley 2011) where players choose which generated image to mutate or evolve from, collectively constructing surprisingly complex images. The algorithms can generate endless novelty with the help of human interactions that provide the missing step for making the system potentially open-ended.

§. https://evocraft.life/
¶. https://gecco-2021.sigevo.org/Competitions

3.5.2 Amorphous computing

The goal of amorphous computing is to study and define the problem of computing with multiple interconnected components that are unreliable, irregular, time-varying, and with limited computational capacity (Abelson et al. 2000). The motivation for this area of research is the development of biological substrates that can compute, function as sensors and actuators, and robustly self-organize to compute with little cost. The precise manufacturing of chemical or biological substrates with the ability to communicate locally through chemical or physical interactions is within reach, making their use as computing vessels particularly attractive. We can embed millions of chips with sensors (Abelson et al. 2000) or program biological cells to serve as logic gates (R. Weiss, G. Homsy, and Nagpal 1998; R. Weiss, G. E. Homsy, and Knight 2002) while relying on cheaper, decentralized parts (Butera 2002).

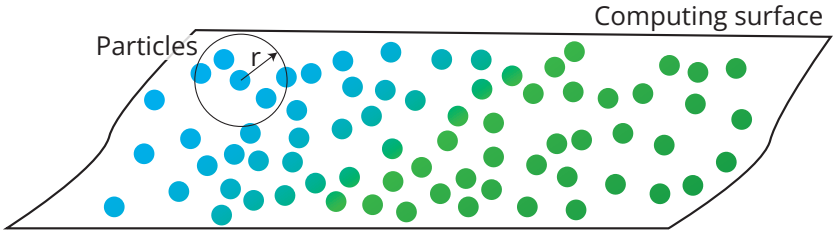


Figure 3.9 – An example of amorphous computing on a surface. Multiple computing particles are arranged randomly. A wave (represented by the color of the cells) propagates within the medium. They can communicate with neighbors within a radius r.

In amorphous computing systems, a medium comprises "computational particles" spread on a surface or mixed in a volume. The particles have limited computing power, with no knowledge of their position or orientation. The particles can communicate with neighbors via some specified mechanism, which may be analogous to biological ones. The two main components of this research are *synthetic biology* and the choice of computing paradigm.

Synthetic biology. The design of a substrate from chemical processes or genetically engineered biological cells (R. Weiss, G. E. Homsy, and Knight 2002). It should behave correctly and follow predefined rules.

Computing paradigm. How to program individual agents to follow a predefined global goal (Nagpal 2001)?

The growing point language (GPL) is an example of a programming language that enables programmers to specify complex patterns for computational particles, which are internally represented in the particles as state machines (Coore 1999).

Nagpal (2002) developed another programming language inspired by biological cell differentiation (Lawrence 1992; Wolpert 1969) that can be compiled into individual agent programs to follow some global specifications.

Chapter

4 Measuring complexity in evolving complex systems

In this chapter, we propose an approach to measure the growth of the complexity of emerging patterns in complex systems such as Cellular automata (CAs). We discuss several ways in which a metric to measure complexity growth can be defined. This includes approaches based on compression algorithms and artificial neural networks. We believe that such a metric can be useful for designing systems that could exhibit open-ended evolution, which itself might be a prerequisite for the development of general artificial intelligence. We conduct experiments on 1D and 2D grid worlds and demonstrate that using the proposed metric, we can automatically construct computational models with emerging properties similar to those found in Conway's Game of Life, as well as many other emergent phenomena. Interestingly, some of the patterns we observe resemble forms of artificial life (Langton 1993). We test our metric of structural complexity growth on cellular automata, but it can also be applied to a wide range of complex systems.

4.1 Introduction

Recent advances in machine learning and deep learning have had success in reproducing some very complex feats traditionally thought to be only achievable by living beings. However, making these systems adaptable and capable of developing and evolving on their own remains a challenge that could be crucial to eventually developing AI with general learning capabilities (for example, as is further discussed in (Mikolov, Joulin, and Baroni 2018)). Building systems that mimic some key aspects of the behavior of existing intelligent organisms (such as the ability to evolve, improve, adapt, etc.) might represent a promising path. Intelligent organisms — for example, human beings but

also most living organisms, if we consider a broad definition of intelligence — are a form of spontaneously occurring, ever-evolving complex systems that exhibit these kinds of properties (Booker 2004). The ability to sustain open-ended evolution appears to be a requirement in order to enable the emergence of arbitrarily complex adaptive systems.

Although a rigorous attempt at defining intelligence or life is beyond the scope of this work, we assume that a system we might identify as evolving, with the potential of developing intelligence, should have the property of self-preservation and the ability to grow in complexity over time. These properties can be observed in living organisms (Booker 2004) and should also be part of computational models that aim to mimic them.

To recognize self-preservation and growth in complexity, one should be able to detect emerging macro-structures composed of smaller elementary components. For the purpose of obtaining computational models that grow in complexity over time, one should also be able to determine the amount of complexity these systems contain. In this chapter, we propose and discuss several ways to estimate complexity and detect the presence of emerging and stable patterns in complex systems such as cellular automata. We show that such metrics are useful when searching the space of cellular automata with the objective of finding those that seem to evolve over time.

4.2 Related work

4.2.1 Artificial life and open-ended evolution

Several works have attempted to artificially create open-ended evolution. A nonexhaustive list of some well-known systems include Tierra (Ray 1991), Sims (Sims 1994), Avida (Ofria and Wilke 2004), Polyworld (Yaeger 1994), Geb (Channon 2003), Division Blocks (Spector, Klein, and Feinstein 2007) and Chromaria (Soros and Stanley 2014). Designs that focus on an objective and make use of reinforcement learning methods to drive evolution are also being studied, e.g. in (Pathak et al. 2019). Most of these simulated "worlds" have had some success reproducing key aspects of evolving artificial life, enabling the emergence of complex behavior from simple organisms. However, they still work within constrained simulated environments and usually consider organisms composed of elementary building blocks, while they do not work outside of this usually

very constrained framework. The divergent and creative evolutionary process could be happening at a much lower conceptual level with fewer assumptions. For this reason, we consider cellular automata in the rest of the chapter because they rely on very few assumptions while offering a very large expressive power and a potentially wide range of behaviors that can be discovered. However, the metrics defined in this work have the potential to be applied to other types of complex system, as discussed in Section 4.7.

4.2.2 Cellular automata

Cellular automata are very simple systems, usually defined in one or two dimensions, composed of cells that can be in a set of states. The cells are updated in discrete time steps using a transition table that defines the next state of a cell given the states of its neighbors. They were originally proposed by Stanislaw Ulam and studied by Von Neumann (Von Neumann and Burks 1966), who was interested in designing a computational system that can self-reproduce itself in a non-trivial way. The trivial self-reproducing patterns were then those that do not have the potential to evolve, for example, the growth of crystals.

Stephen Wolfram later took a more bottom-up approach, beginning with the study of simple 1D binary cellular automata (CA), and identifying four qualitative classes of cellular automaton behavior (Wolfram 1984):

Class 1 evolves to a homogeneous state.

Class 2 evolves to simple periodic patterns.

Class 3 produces aperiodic disordered patterns.

Class 4 produces complex aperiodic and localized structures, including propagating structures.

Wolfram and his colleagues also studied 2D CA using tools from information theory and dynamical systems theory, describing the global properties of these systems in terms of entropies and Lyapunov exponents (Norman H Packard and Wolfram 1985).

Christopher Langton and colleagues also studied CA dynamics (Li, Norman H. Packard, and Langton 1990) — e.g. using the λ parameter (Langton 1990) — and designed a self-replicating pattern much simpler than Von Neumann's (Langton 1984),

now known as Langton loops. The main issues with his system and Von Neumann's universal replicator is the fact that they are very fragile and based on a large amount of human design. As a consequence, although they self-replicate, they cannot increase in complexity and are not robust to perturbations or unexpected interactions with the environment.

A genetic algorithm-based search for spontaneously occurring self-replicating patterns was carried out in 2D cellular automata with several states was undertaken in (Bilotta and Pantano 2011) using entropy measures of the frequency distribution of 3×3 patterns.

4.2.3 Compression and complexity

Compression has often been used as a measure of complexity. Lempel and Ziv have introduced in (Lempel and Ziv 1976) the now widely used Lempel-Ziv (LZ) algorithm as a method for measuring the complexity of a sequence. By constructing back-references to previous parts of a string, the LZ algorithm is capable of taking advantage of duplicate patterns in the input to reduce its size. The DEFLATE algorithm that we use in the following section combines LZ with Huffman coding for an efficient representation of the symbols obtained after the first step. It is one of the most widespread compression algorithms and is, for instance, used in gzip and PNG file compression standards.

The PAQ compression algorithm series (Mahoney 2000) is an ensemble of compression algorithms initially developed by Matt Mahoney with state-of-the-art compression ratio on several compression benchmarks. Better compression of an input means a better approximation of the minimum description length and implicit understanding of more of the underlying patterns in input data. The usefulness of a better compressor is that it can detect much more complex and intricate patterns that aren't simple repetitions of previous patterns.

In (Zenil 2010), H. Zenil investigates the effects of a compression-based metric to classify cellular automata and observes that it results in a partitioning of the space of 1D CA into several clusters that match Wolfram's classes of automata. He also used this approach in the output of simple Turing machines and some 1D automata with more than two states and larger neighborhoods. Extensions of this work include an asymptotic sensitivity analysis of compressed length for input configurations of increasing complexity,

as introduced in (Zenil 2014; Zenil and Villarreal-Zapata 2013).

Additionally, the image decompression time as an approximation of Bennet's logical depth (C. H. Bennett 1995; Zenil, Delahaye, and Gaucherel 2012) and the output distribution of simple Turing machines combined with block decomposition of CA to approximate their algorithmic complexity have also been investigated (Soler-Toscano et al. 2014; Zenil, Soler-Toscano, et al. 2015). However, the possible extent to which such measures of complexity could be applied to more complex automata and other complex systems has not yet been extensively studied. For a review of several measures of complexity and their applications, see (Grassberger 1989b).

4.3 Compression-based metric

A cellular automaton of size n in 1D can be represented at time t by its grid state $S^{(t)} = \{c_1^{(t)}, ..., c_n^{(t)}\}$ where each c_i (also called a cell) can take one of k possible values (representing the possible states) and a transition rule ϕ. The transition rule is defined with respect to a neighborhood radius r with the mapping $\phi(c_{i-r}^{(t)}, ..., c_i^{(t)} ..., c_{i+r}^{(t)}) = c_i^{(t+1)}$ that maps $\{1, ..., k\}^{2r+1}$ to $\{1, ..., k\}$. The quantity $2r + 1$ corresponds to the number of cells taken into account to calculate the next state of a cell, namely the cell itself and the r neighboring cells in both directions.

Simulating a CA amounts to the recursive application of this mapping ϕ to an initial state $S^{(0)} = \{c_1^{(0)}, ..., c_n^{(0)}\}$.

In the rest of the chapter, we consider cyclic boundary conditions for the automata, meaning that the indices $i - r, ..., i + r$ above are taken modulo n the size of the automaton in 1D. Boundary conditions can have some effect on a CA's evolution, but cyclic boundaries have been empirically observed to have limited effect on the complexity of automata in 1D (LuValle 2019).

The definition given in the equation above can be extended to higher dimensional automata by modifying the neighborhood and the definition of ϕ. A 2D neighborhood of radius 1 can be defined as the 3 by 3 square around the center cell — also called the Moore neighborhood — or by only considering the four immediate horizontal and vertical neighbors of the center cell — the Von Neumann neighborhood.

Elementary cellular automata (ECA) are 1D CA with $k = 2$ and $r = 1$. There are

2^3 elements in $\{1, ..., k\}^{2r+1}$ and $2^{2^3} = 256$ possible different set transition rules that are often compactly represented as a binary string with 8 bits. The relatively low number of rules of this type makes it possible to appreciate the performance of a metric and compare it with others.

We define the compressed length C of a 1D cellular automaton at time t as

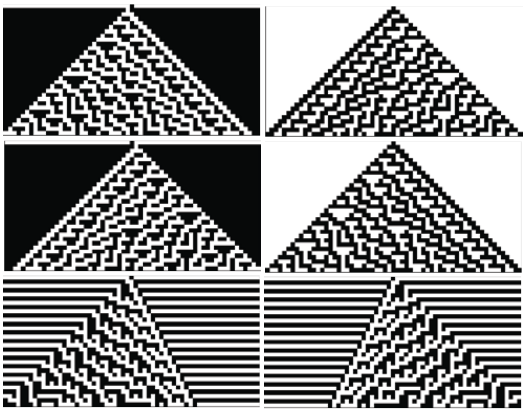
$$C(S^T) = \text{length}\left(\texttt{comp}(c_1 \mid\mid c_2 \mid\mid ... \; c_n)\right) \tag{4.1}$$

where $||$ denotes the string concatenation operator, and cells c_i are implicitly converted into string characters (with one symbol per unique state). `comp` is a compression algorithm that takes a string as input and outputs a compressed string, and length is the operator that returns the length of an input string.

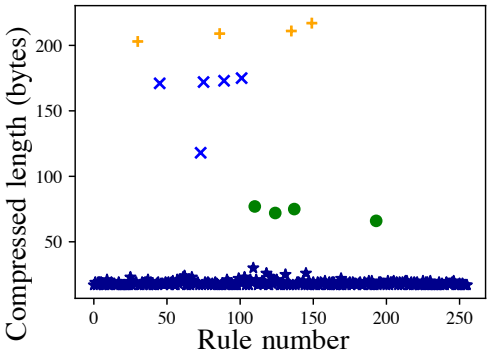
Similarly to (Kowaliw 2008; Zenil 2010), we use zlib's C implementation of DEFLATE to compress the final state of the automaton. If we apply the above metric to the 256 ECA run for 512 timesteps and initialized with one activated cell in the middle, we obtain the plot of Figure 4.1b. This example is re-used in the rest of the chapter as a way to easily visualize and check that the defined complexity measures are coherent with one another. The colors in Figure 4.1b were obtained with a KMeans clustering algorithm applied on the compressed length of the automata states.

As visible on Figure 4.1b, rules are clearly separated into several clusters that turn out to match Wolfram's classification of ECA. Class 3 behavior can be found at the top of the plot (highest compressed length, orange and blue clusters), Class 1 and 2 are clearly separated at the bottom part (not detailed here) and Class 4 rules (colored in green) lie in between the other types of behavior. The 6 highest scoring rules are shown on Figure 4.1a and correspond to Class 3 behavior in Wolfram's classification. Among the classes of behavior, some sub-clusters can be found that correspond to similarly behaving rules.

Ultimately, the theoretical goal of using compression algorithms is to approach the theoretical minimum description length of the input (Grunwald 2007). For very regular inputs, this length should be relatively small and inversely for random inputs. However, gzip and PAQ are crude approximations of the minimum description length and may



(a) 6 highest scoring automata. Only the first 30 timesteps are shown for readability.



(b) All 256 compressed length scores.

Figure 4.1 – Compression-based metric on 1D ECA. 4.1a represents the 1D ECA evolution with each line being the state of the automaton at a given timestep, starting from a single cell set to 1. Cells which are in state 1 are represented in black and cells in state 0 are represented in white. Time increases downwards. Figure 4.1b represents the compressed length of the 256 ECA rules, with different marker and colors corresponding to the obtained KMeans clusters.

only approach it in a given context. As an example, compressing text data (a task often performed with gzip in practice) is much more efficient with a language model that can assign a very low probability to non grammatically correct sentences. The Kolmogorov complexity (Kolmogorov 1968) of a cellular automaton is upper bounded by a value that is independent of the chosen rule, as it is entirely determined by the transition table, the grid size, initial configuration and number of steps.

4.4 Predictor-based metric

One obvious limit of using compression length as a proxy for complexity is the fact that interesting systems mostly have intermediate compressed length. Compressed length increases with the amount of disorder in the string being compressed. Therefore, extreme lengths correspond either to systems that do not increase in complexity on the lower end of the spectrum, or systems that produce a maximal amount of disorder on the higher end. Neither of them have the potential to create interesting behavior and increase in complexity. Intermediate values of compressed length are also hard to interpret, since average lengths might either correspond to interesting rules or slowly growing disordered systems.

To cope with this limitation, one should also take into account the dynamics of complexity, that is, how the system builds on its complexity at a given time as it keeps evolving, while retaining some of the structures it had acquired earlier. Compression leverages the amount of repetitions in inputs to further compress, and this may also be used as an estimate of structure overlap, as explained in the following section.

4.4.1 Joint compression

As a way to both measure the complexity and the amount of overlap between two automata states apart in time, we define a joint compressed length metric for a delay τ as

$$C'\left(S^{(T+\tau)}, S^{(T)}\right) = C\left(S^{(T)} \mid\mid S^{(T+\tau)}\right) \tag{4.2}$$

where $||$ represents the concatenation operator. This quantity is simply the compressed length of a pair of global states — defined at the beginning of 4.3, represented by the letter S — at two timesteps separated by a delay τ. In 1D, concatenation means chaining the two string representations before compressing, and in 2D we can chain two flattened representations of the 2D grid. This introduces several issues which we discuss in Section 4.4.2.

To quantify the amount of overlap between the two global states, we can compute the ratio of this joint compressed length with the sum of the two compressed lengths $C(S^t)$ and $C(S^{t-\tau})$, thereby forming the joint compression score

$$\mu = \frac{C\left(S^t\right) + C\left(S^{t-\tau}\right)}{C'\left(S^t, S^{t-\tau}\right)} \tag{4.3}$$

defined for an automaton S, time t and delay τ.

This metric is based on the intuition that if patterns occur at step $T - \tau$ of the automaton's evolution and are still present at step T, the joint compressed length will be lower than the sum of the two compressed length. The idea is illustrated in Figure 4.2, where it is pointed out that a stable moving structure (sometimes called *glider* or *spaceships* in Game of Life) will yield lower joint compressed lengths. This also applies to structures that self-replicate, grow from a stable seed or maintain the presence of

some sub-structures. Bigger structures yield a higher compression gain.

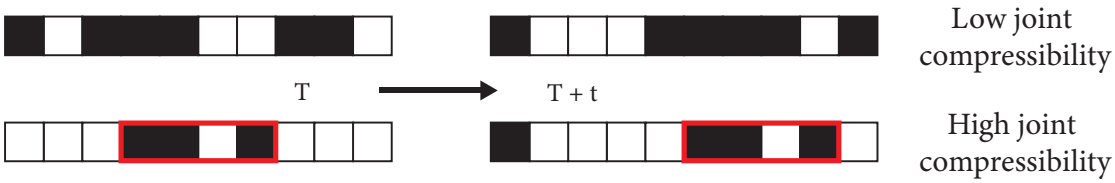
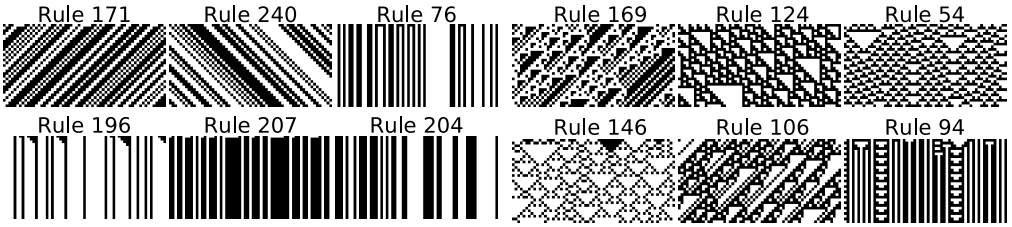


Figure 4.2 – Joint compression method illustration. If a structure persists through time, this will decrease the joint compressed length compared to the sum of compressed lengths. A persistent structure is circled in red.

Joint compression alone is not sufficient since it only selects rules that either behave like identity or shift input because they maximize the conservation of structures through time — as illustrated in Figure 4.3a. However, one may combine the joint compression score with another complexity measure to only select rules that exhibit some disorder, or growth in complexity — as Figure 4.3b shows (the condition here was a threshold on the difference of compressed length between initial and final states).



(a) Highest joint compression score among the 256 ECA.

(b) With condition on compressed length increase.

Figure 4.3 – Comparison of the raw joint compression score and the addition of a complexity increase condition. The high overlap in structures is not enough to get interesting rules a shown in 4.3a, but the addition of a complexity threshold allows to retrieve rules with complex but still structured behavior, as shown in 4.3b. Figures are from the same slice of 60 cells over 30 timesteps taken from larger automata with random initial states. The top row corresponds to $t = 0$ and time increases downwards.

4.4.2 Count-based predictor

A major issue with the joint compression metric is the fact that it is designed to compress a linear stream of data. This is not ideal when considering higher dimensional automata. Larger sets of transformations have to be considered such as translations, rotations, flips, etc. Theoretically this should not be a problem for a good enough linear compression algorithm, but hardware and software limitations make it impractical to

work with existing algorithms on higher dimensional structures — with e.g DEFLATE's upper limit on dictionary size.

These higher dimensional automata might be better at generating complex dynamics, and the large size of their rule spaces makes it a challenge to explore. There has been at least one attempt to deal with these higher dimensional systems (Zenil, Soler-Toscano, et al. 2015) that lacks the scalability to work with large inputs.

An alternative to the linear compression-based method presented above would be to use compressors optimized for n-dimensional data (e.g. PNG compression for 2D automata) to take advantage of spatial correlation for compressing. However, these compressors are rare for higher dimensional data, and are usually optimized for one type of input — e.g. images with PNG.

Another way to tackle the problem is to use a prediction based approach to compression. Similarly to methods described in (Schmidhuber and Heil 1996) and one of the first steps of the PAQ compression algorithm (Mahoney 2000), we learn a statistical model of input data to predict the content of a cell given its immediate predecessors. For compression, this is often followed by an encoding step — using Huffman or arithmetic coding — that encodes data which contains the least information (least "surprising" data) with the most compact representation. This approach can also be related to the texture synthesis method described in (Efros and Leung 1999), where the authors learn a non parametric model to predict the next pixel of a texture given a previously synthesized neighborhood. Additionally, because we do not need the operation to be reversible as in regular compression, it is not necessary to limit the prediction model to making prediction with predecessors only.

For a global state $S = (c_1, ...c_i, ..., c_n)$, the neighborhood of cell i with radius r, denoted $n_{r,i}$ is defined as the tuple $n_{r,i} = (c_{i-r}, ...c_{i-1}, c_{i+1}..., c_{i+r})$ — without the middle cell. The goal of this method is to estimate the conditional probability distribution $p(s|n_r)$ of the middle states at timestep T given a neighborhood of radius r. Assuming cell states given their neighborhood can be modeled by mutually independent random variables, the log-probability of global state $S^{(T)}$ is written

$$\log p(S^{(T)}) = \log \prod_{i=1}^{N} p(c_i|n_{r,i}) = \sum_{i=1}^{N} \log p(c_i|n_{r,i}) \tag{4.4}$$

If the automaton has a very ordered behavior, a model will predict with high confidence the state of the middle cell given a particular neighborhood. On the other hand, in the presence of maximal disorder, the middle cell will have an equal probability of being in every state no matter the neighborhood. In the latter case, a predictive model minimizing $-\log p(S^{(T)})$ would yield a high negative log-probability.

A simple possible predictor for such purpose is a large lookup table that maps all visited neighborhoods to a probability distribution over the states that the middle cell can be in. State distributions for each neighborhood are obtained by measuring the frequency of cell states given some observed neighborhoods. We denote by Λ this lookup table, defined for a window of radius r, which maps all possible neighborhoods of size $2r+1$ (ignoring the middle cell) to a set of probabilities p over the possible states $\{s_1, ..., s_n\}$, and p can be written $[p_{s_1}, p_{s_2}, ..., p_{s_n}]$. Λ is defined by

$$\begin{aligned} \Lambda : \quad \{s_1, ..., s_n\}^{2r} &\to \quad \Delta_n \\ n_{r,i} &\mapsto \quad p \end{aligned} \tag{4.5}$$

where Δ_n denotes the probability simplex in dimension n.

To measure the uncertainty of that predictor, we can compute the cross-entropy loss between the data distribution it was trained on and its output. We compute the log probability of the observed data given the model, or the quantity

$$L = -\frac{1}{N}\sum_{i=1}^{N}\sum_{k=1}^{n} \mathbb{1}_{\{s_k\}}(c_i) \log \Lambda(n_{r,i})_{s_k} \tag{4.6}$$

where $\mathbb{1}_{\{s_k\}}$ denotes the indicator function of the singleton set $\{s_k\}$. An illustration of the counting process is represented in Figure 4.4. The quantity L is minimal when the $\Lambda(n_{r,i})_{s_k}$ is always equal to one, which means that the state of every cell is entirely determined by its neighborhood.

We apply this metric to all 256 ECA, with a window radius of size 3 (the 6 closest neighbors are used for prediction), and the same settings as for Figure 4.1b. Cross-entropy loss of the lookup table gives the results of Figure 4.5a. Colors are the same as in Figure 4.1b for comparison purposes.

We note the similarity between this plot and the one from Figure 4.1b, with a roughly

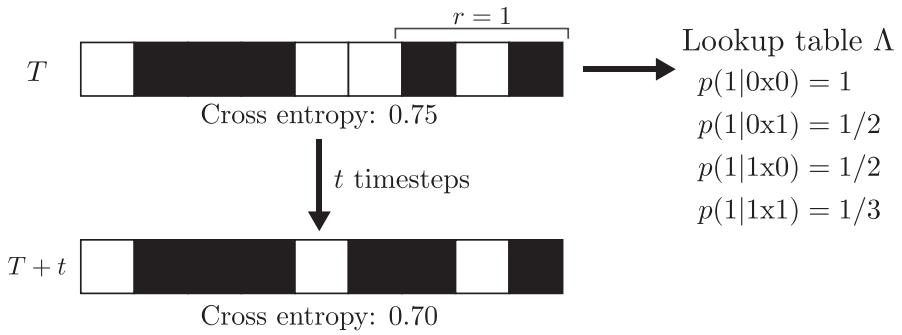
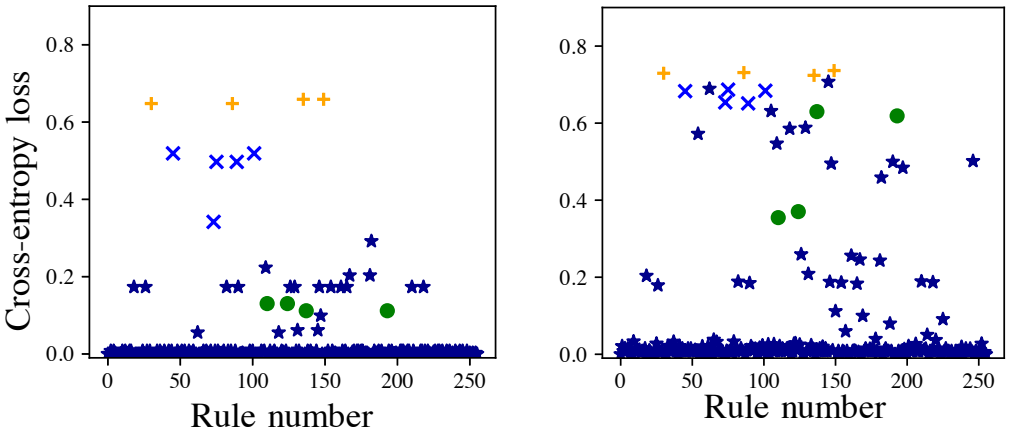


Figure 4.4 – Count-based predictor method for a radius $r = 1$. A frequency lookup table is computed from the global state at time T by considering all neighborhoods with radius $r = 1$ (3 consecutive cells but ignoring the middle cell). Cross-entropy with the automaton at time T quantifies the overall complexity. This can be compared to the cross-entropy at time $T + t$ for the amount of overlap.



(a) Count-based predictor **(b)** Neural network-based predictor

Figure 4.5 – Average cross entropy loss for the two predictor-based methods on all 256 ECA. Rules are separated in several clusters. The count-based predictor (left plot) and neural network-based predictor (right plot) were applied with a neighborhood radius $r = 1$ and 3.

equivalent resulting classification of ECA rules, with the exception of rules with low score. Rules that produce highly disordered patterns are on top of the plot whereas the very simply behaving rules are at the bottom. This indicates coherence between the two metrics.

4.4.3 Neural network based predictor

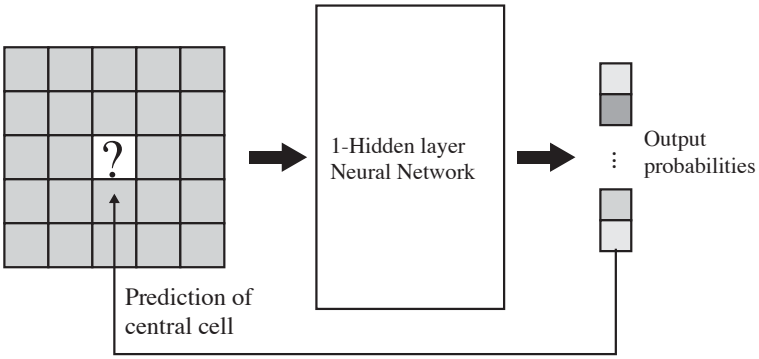


Figure 4.6 – Neural network architecture for predicting a central cell given its neighbors. Output probabilities are defined for all possible states of the central cell.

The frequency-based predictor described above still has limitations:

— It does not take into account any redundancy in the input which may lead to suboptimal predictions (in a CA, very similar positions might have similar center cell state distribution, e.g. a glider in Game of Life should be recognized by the model no matter the rest of the neighborhood).

— For the same reasons, when considering large window sizes, the number of possible neighborhood configurations grows much larger than the number of observed ones, leading to an input sparsity problem.

More sophisticated models can cope with above limitations by dealing with high-dimensional inputs without sparsity problems, and taking into account redundancy of inputs and potential interactions between states for prediction.

We measure the cross-entropy loss of this simple model on the training set after a standard learning procedure, which is the same for all rules. The procedure is applied to a one-hidden-layer neural network with a fixed hidden-layer size. We use a ReLU nonlinearity for the hidden layer and a softmax to obtain the output probabilities.

For n possible states $s_1, ..., s_n$, a cell in state s_k is represented as a vector of 0s of

size n with a 1 in position k. The input to the network is the concatenation of these cell vectors for all cells in the neighborhood. The output of the network is a vector of size n with the output probability for each state.

Gradient updates are computed during training to minimize the cross-entropy loss between outputs and target examples. For a timestep T, we use the training procedure in order to minimize with respect to θ the following quantity,

$$L_\theta^{(T)} = -\frac{1}{N}\sum_{i=1}^{N}\sum_{k=1}^{n} \mathbb{1}_{\{s_k\}}\left(c_i^{(T)}\right) \log\left[f_\theta\left(n_{r,i}^{(T)}\right)_{s_k}\right] \tag{4.7}$$

where the neural network depending on parameter θ is denoted f_θ, and $n_{r,i}^{(T)}$, the neighborhood of cell i with radius r at time T is defined in the same way as in eq. equation 4.6. Loss is calculated with respect to the testing set at time $T+\tau$ by computing the same quantity at this subsequent timestep.

The training procedure is selected to achieve reasonable convergence of the loss for the tested examples. It must be well defined and stay the same to allow for comparison of the results across several rules. The score in timestep T for a delay τ is computed with the following formula

$$\mu_\tau = \frac{L^{(T)}}{L^{(T+\tau)}} \tag{4.8}$$

where $L^{(T+\tau)}$ is the log probability of the automaton state at timestep $T+\tau$ (defined in eq. equation 4.7) according to a model with parameters learned during training at timestep T and $L^{(T)}$ is the same as in eq. equation 4.7. The value μ_τ will be lower for easily "learnable" global states that do not translate well in future steps — they create more complexity or disorder — thereby discarding slowly growing very disordered structures. Higher values of μ_τ correspond to automata that have a disordered global state at time T that can be transposed to future timesteps relatively well. Those rules will tend to have interesting spatial properties — not trivially simple but not completely disordered because the model transposes well — as well as a large amount of overlap between a given step and the future ones, indicating persistence of the spatial properties from one state to another. We also selected the metric among other quantities computed from $L^{(T)}$ and $L^{(T+\tau)}$ because it gave the best score in our experimental datasets.

4.5 Experiments

We carried out several experiments on a dataset of 500 randomly generated 3 states ($n = 3$) rules with radius $r = 1$. Those rules were manually annotated for interestingness, defined as the presence of visually detectable non-trivial structures. The annotation was carried out by five persons, using a majority rule to assign a binary label to each rule. The dataset contains 46 rules labeled as interesting and 454 uninteresting rules. Ranking those rules with the metrics introduced above allows us to study the influence of parameters and the adequacy between interestingness as we understand it and what the metric measures.

The task of finding interesting rules can be framed as either a classification problem or a ranking problem with respect to the score we compute on the dataset. The performance of our metric can be measured with the usual evaluation metrics used on these problems, and, notably, the average precision (AP) of the resulting classifier.

The average precision scores for the neural network and the count-based methods for time windows of 5, 50, and 300 timesteps are given in Table 4.1. Scores were calculated in automata of size 256×256 cells, ran for 1000 timesteps ($T + \tau = 1000$). Scores were computed for radii ranging from 1 cell (8 nearest neighbors) to 5 cells (120 neighbors), with a one-layer neural network containing 10 hidden units trained for 30 epochs with batches of 8 examples. The best AP for each time window is shown in bold. Results for the frequency lookup table predictor are only shown for $r = 1, 2$ because of the sparsity issues with the lookup table from $r = 2$ and above, making it unpractical to use the table — 3^{24} possible entries for the lookup table with $r = 2$ against only 256^2 observed states.

From these experiments, larger radii appear to perform slightly better, although not in a radical way. Since the number of neighbors scales with the square of the radius, reasonably small radii might be a good trade-off between performance and computational cost of the metric.

We also study the performance of our metrics — lookup table and neural network-based — as inputs of a binary classifier against two simple baselines on a random 70/30 split of our dataset. The first baseline classifies all examples as negative. The second baseline is based on compressed length as defined in (Zenil 2010) and computed by

Table 4.1 – Experimental results - AP scores

Neural network	$r = 1$	2	3	4	5
5 steps	0.387	0.448	**0.541**	0.525	0.534
50 steps	0.377	0.433	0.517	0.491	**0.542**
300 steps	0.358	0.454	0.488	**0.527**	0.525
Lookup table	$r = 1$	2			
5 steps	0.092	0.070			
50 steps	0.102	0.070			
300 steps	0.093	0.069			

This table shows the average precision (AP) scores obtained on the dataset of section 4.5 with the neural network-based and lookup table-based methods. Results are shown for delays $\tau = 5, 50, 300$ and several radii values r.

Metric	Baseline	Compressed length (Zenil 2010)	Lookup Table	Neural network
Accuracy	0.90	0.913	0.926	**0.953**

Table 4.2 – Experimental results - Accuracy of each metric of complexity when used to classify which automatons do evolve interestingly, compared against the trivial all-negative baseline and the compressed length metric (Zenil 2010).

choosing a pair of thresholds that minimize mean square error when classifying examples in between as positive — this is based on the observation made in Section 4.3 that interesting rules have intermediate compressed lengths. Results are in Table 4.2 where only the best radius is shown. The lookup table performs better than the baselines, but the neural network gives the best score.

Above experiments demonstrate the capability of our proposed metric to match a subjective notion of interestingness of our labeling. For instance, the top 5 and top 10 scoring rules of the best performing configuration ($r = 3$, $\tau = 5$) are all labeled as interesting, and top 100 scores contain 41 of the 46 rules labeled as interesting.

4.6 Discussion

In this section, we discuss the results obtained by using the metric of equation 4.8 and the way they can be interpreted.

One dimensional cellular automata By applying the metric on the same example as before, we again obtain a plot with a rule classification that matches a visual appreciation of complexity of 1D CA. Results are shown on Figure 4.5b. Similarly to the previous cases, rules we might label as interesting are unlikely to be either at the top or bottom of the plot.

Two dimensional cellular automata Simulations conducted with 2D CA used grids of size 256×256. Automata were ran for 1000 steps (the metric is measured with respect to the reference time $T = 1000$). Rules are defined with a table of transitions from all possible neighborhood configurations with radius $r = 1$ (3×3 squares) to a new state for the central cell. Unbiased sampling of rules, obtained by uniformly sampling the resulting state for each transition independently, overwhelmingly produces rules with a similar amount of transitions towards each state and fails to produce rules without completely disordered behavior more than 99% of the time.

Therefore, we adopt a biased sampling strategy of the rules, selecting the proportion of transitions towards each state uniformly on the simplex — e.g for 3 states we might get the triple $(0.1, 0.5, 0.4)$ and sample transitions according to these proportions. This parametrization can be related to Langton's lambda parameter (Langton 1990) that takes into account the proportion of transitions towards a transient (inactive) state and all the other states. We obtain approximately 10% interesting rules with this sampling as the proportions of our experimental dataset show.

Using the neural network-based complexity metric, we were able to find rules with interesting behavior among a very large set through random sampling. Some of these rules are shown here. Figure 4.8 displays three 2D rules that were selected manually upon visual inspection among the 20 highest scoring for metric μ_{50} (defined in eq. equation 4.8) of a sample of 1700 randomly generated 2-states 3 by 3 neighborhood rules. For comparison, Conway's Game of Life rule (GoL) ranks in the top 1% of the 2500 rules mentioned above for runs that do not end in a static global state. We observe that spontaneous glider formation events appear to be captured by our metric. Although gliders in cellular automata are a simple process that can be created manually, detection of their spontaneous emergence within a random search setting is a first step towards

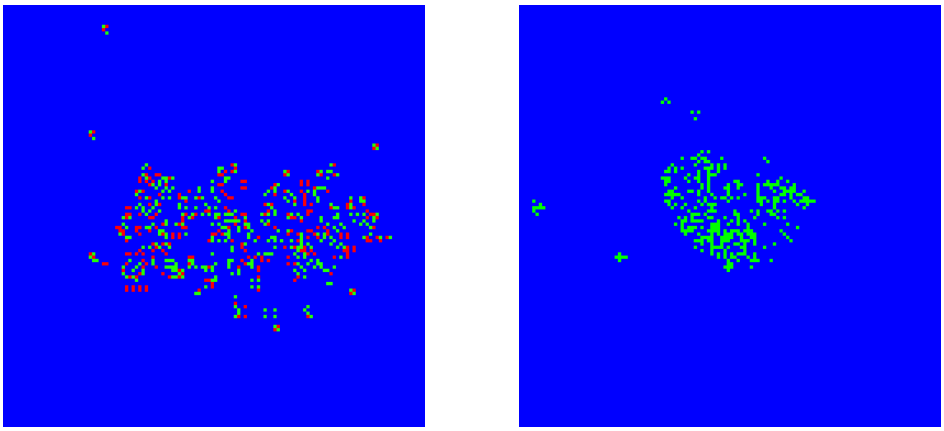


Figure 4.7 – Rules with 3 states that have spontaneously occurring glider structures. The gliders are the small structures that are outside of the center disordered zone. Some of them move along the diagonals while some others follow horizontal or vertical paths. Note that some repeating patterns occur also in the more disordered center zone.

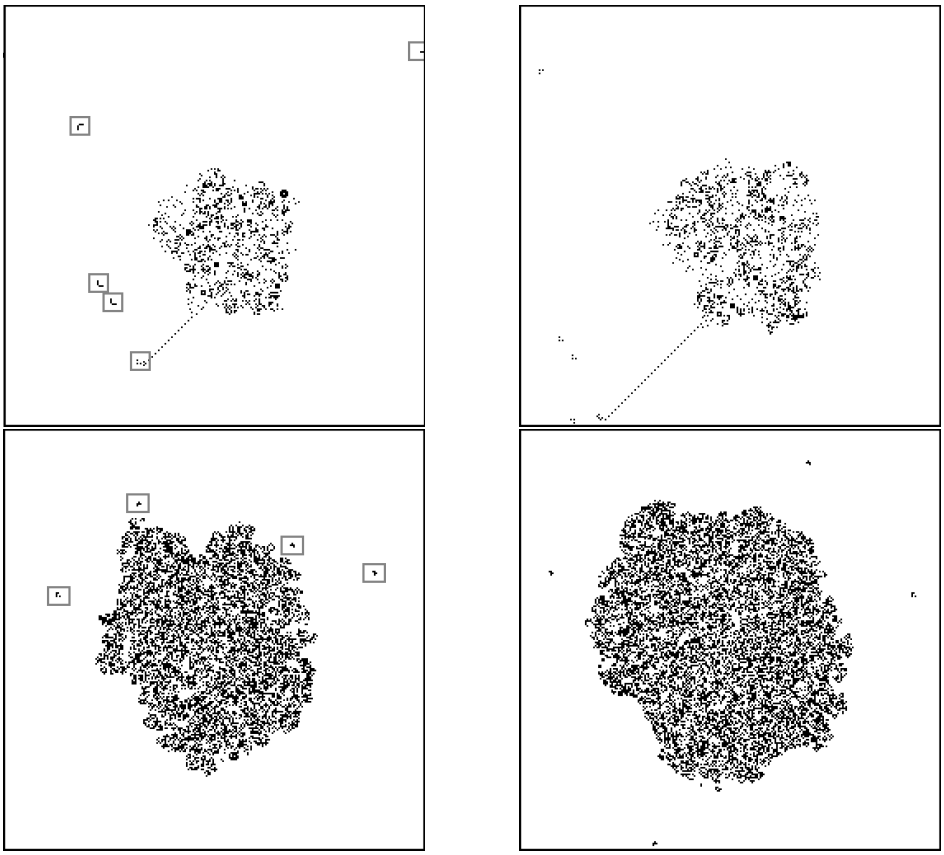
finding more complex macro structures that can emerge out of simple components. Rules with low scores are overwhelmingly of the disordered kind.

Figures 4.7, 4.9 and 4.10 show some three states rules that were selected through random sampling on the simplex with the neural network-based metric. They were selected among the 30 highest scoring rules out of 2500 randomly selected 3 states rules. All of their behaviors involve the growth and interaction of some small structures made of elementary cells.

All automata were initialized with a random disordered square of 20 by 20 cells in the center. In the Figures mentioned above, colors were normalized with the most common state set to blue. Figure 4.7 shows rules that spontaneously emit gliders that go through space in a direction until they interact with some other active part of the automaton. Figure 4.9 shows rules that generate small structures of between four and thirty cells that are relatively stable and interact with each other. These elementary components could be a basis for the spontaneous construction of more complex and bigger components. Figure 4.10 shows some other rules from this set of high ranking automata. They highlight the wide range of behaviors that can be obtained with these systems. Interesting rules from our search process can be found, along with other examples, in the form of animated GIFs *.

For some of these rules interesting patterns appear less frequently in smaller grids,

*. https://bit.ly/interesting_automata



(a) Timestep T **(b)** Timestep $T + 50$

Figure 4.8 – Spontaneous glider formation and evolution is observed for some high scoring 2 states rules. Each row corresponds to a rule, with a 50 timesteps difference between the two columns. Gliders are marked with a gray square. Runs were initialized with a small 20 by 20 disordered square (uniformly sampled among possible configuration) in the center simulated for up to 400 steps.

indicating that the size of the space might impact the ability to generate complex macro-structures. Increasing the size of the state space to very large grids might therefore make it easier to generate very complex patterns.

4.7 Conclusion

In this chapter, we have proposed compression-inspired metrics for measuring a type of complexity occurring in complex systems. We demonstrated its usefulness for selecting CA rules that generate interesting emergent structures from very large sets of possible rules. In higher dimensions where linear compression, as in gzip, is not sufficient to find complex patterns, our metric is also useful.

We study 2 and 3 states automata here and we plan to investigate the effects of

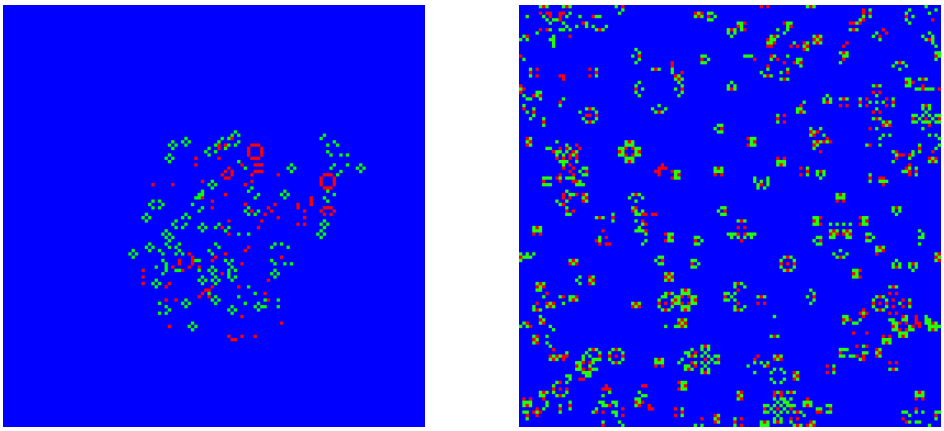


Figure 4.9 – Rules with 3 states that generate cell-like interacting structures. These patterns are either static or moving and can interact with one another to generate copies of themselves and other patterns. Note the very similar micro-structures that are repeated at several places in the space.

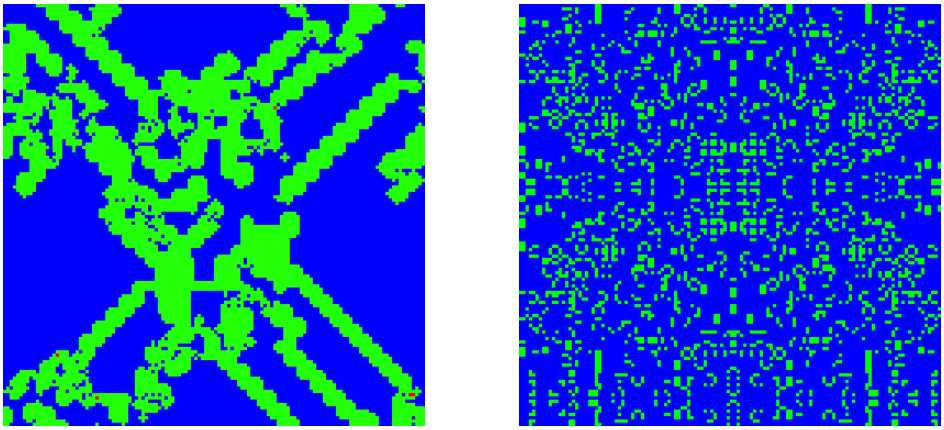


Figure 4.10 – Rules with surprising behaviors that are highly structured but complex. Those rules were selected among high-ranking rules for the neural-network based complexity metric. They all exhibit structurally non trivial behavior.

additional states or larger neighborhoods on the ability to evolve more structures and obtain more interesting behaviors.

The dataset and code to reproduce our experiments and improvement on the reported results are published on GitHub [†]. The metrics we introduce in this chapter could be used to design organized systems of artificial developing organisms that grow in complexity through an evolutionary mechanism. A possible path toward such systems could start by creating an environment where computational resource allocation favors the fraction of subsystems that perform the best according to our measure of complexity.

The proposed metric is theoretically applicable to any complex system where a

†. https://github.com/hugcis/evolving-structures-in-complex-systems

notion of state of an elementary component and locality can be defined. With these requirements satisfied, we can build a similar prediction model that uses information about local neighbors to predict the state of a component, and thereby assess the structural complexity of an input.

We believe that the capability of creating evolving systems out of such elementary components and with few assumptions could be a step towards AGI. By devising ways to guide this evolution in a direction that we find useful, we would be able to find an efficient solution to hard problems while retaining the adaptability of the system. It might be suitable to avoid over-specialization that can happen in systems designed to solve a particular task — e.g. reinforcement learning algorithms that can play games, and supervised learning — by staying away from any sort of objective function to optimize and by leaving room for open-ended evolution.

Chapter

5 Visualization of computations in large-scale complex systems

Emergent processes in complex systems such as cellular automata can perform computations of increasing complexity, and could possibly lead to artificial evolution. Such a feat would require scaling up current simulation sizes to allow for enough computational capacity. Understanding complex computations happening in cellular automata and other systems capable of emergence poses many challenges, especially in large-scale systems. We propose methods for coarse-graining cellular automata based on frequency analysis of cell states, clustering and autoencoders. These innovative techniques facilitate the discovery of large-scale structure formation and complexity analysis in those systems. They emphasize interesting behaviors in elementary cellular automata while filtering out background patterns. Moreover, our methods reduce large 2D automata to smaller sizes and enable identifying systems that behave interestingly at multiple scales.

5.1 Introduction

Cellular automata (CA) have been extensively studied since the 1960s. Originally designed and studied to create artificial evolution from self-replication (Langton 1984; Von Neumann and Burks 1966), previously studied cellular automata simulations were often of relatively modest sizes. Only specific rules with repetitive or predictable dynamics such as John Conway's Game of Life (Gardner 1970) have been scaled up to larger grid sizes ($10^4 \times 10^4$ or more cells).

For complex phenomena such as artificial evolution to exist and be open-ended within

those simulated worlds, there needs to be sufficient "capacity" — a large enough state-space. In nature, complex and significantly different dynamics often arise from uniform laws at a smaller scale (Anderson 1972). It seems unlikely that such complex processes, like artificial evolution, could happen in too small CAs because higher order dynamics do not have enough capacity to emerge. However, several issues arise when scaling CAs to large sizes:

— Time complexity rapidly becomes a bottleneck. Updating a large number of cells is costly. Tricks such as caching of some of the computations can help, but do not always improve performance significantly (Gosper 1984).

— Memory complexity can also become an issue when dealing with numerous states, and especially grids in 3 dimensions and more. In that case, even the underlying rule of the system cannot be stored within reasonable memory capacity.

— Visual inspection of these large grids is infeasible. Studying CA complexity is rendered difficult by the highly variable nature of emergent processes. It is especially the case for large-scale systems.

When working with such large systems, it is less relevant to focus on the local behaviors at the single cell level. This is similar to other complex systems like the weather, in which behaviors of individual atoms in a cloud are irrelevant to large-scale air mass movements. Much richer behaviors can be observed from studying large patterns' formation and their evolution. This should also hold true for CAs; we further discuss this question in Conclusion.

In this chapter, we investigate techniques that can help us visualize large space-time diagrams of CAs. We demonstrate that simple clustering and coarse-graining techniques can be used in order to perceive structures that cannot emerge on smaller grids. This is also useful for disordered cellular automata with hidden structures as is the case for the elementary cellular automaton rule 18, illustrated in Figure 5.1 — more details in Results.

Reducing large grids to smaller sizes while preserving interesting behaviors such as pattern formation is essential to apply to these CAs complexity metrics designed to work on modestly sized grids (Grassberger 1986; Soler-Toscano et al. 2014; Zenil 2010; Zenil, Soler-Toscano, et al. 2015). Common metrics of complexity are often limited by the

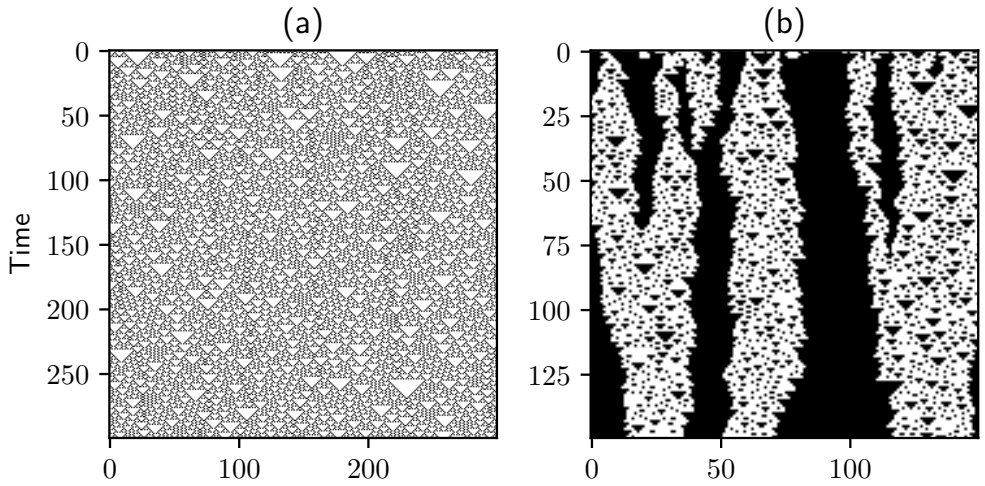


Figure 5.1 – Hidden structures in rule 18 are uncovered by filtering the space-time diagram with our frequency histogram-based method. **(a)** shows 300 timesteps of a randomly initialized rule 18 simulation. Notice the complex structures made visible in **(b)** with our method.

number of components in the systems (number of cells in a CA grid, timesteps, etc.) or may not be effective when small-scale patterns are less relevant than large-scale ones.

5.2 Related work

Previous work on coarse-graining cellular automata focused either on conserving the main computational properties of CA rules through exact coarse-graining or on filtering interesting behaviors without reducing the amount of computations. Our work both highlights interesting behaviors and compresses the representation, which we argue are necessary to study complexity in large cellular automata.

5.2.1 Coarse-graining in cellular automata

Coarse-graining is an approximation procedure used to speed up computations in systems made of many components. It originated in (Levitt and Warshel 1975) and is now widely used in physics to model complex systems at various granularity levels and is successful at modeling bio-molecules (Ingólfsson et al. 2014; Kmiecik et al. 2016; Potoyan, Savelyev, and Papoian 2013).

The exact coarse-graining of elementary cellular automata (ECA) has been investigated extensively in (Israeli and Goldenfeld 2004, 2006). Authors found ways of rewriting

one-dimensional CA rules into each other through coarse-graining of the transition rule. They built a graph of equivalence of all 256 ECA and identified some rules that do not admit any computational reduction. This indicates that some cellular automata are accomplishing fundamentally more computations than others.

5.2.2 Filtering

Filtering cellular automata (CA) was introduced to reduce a CA's behavior to its most relevant parts. The goal is to extract relevant irregularities from a CA's space-time diagram. Seminal work by (Hanson and James P. Crutchfield 1992, 1997) formalized the notion of domains and coherent structures in cellular automata. They used a set of regular languages to represent cellular automata dynamics and extract relevant behaviors such as discontinuities between regular domains or "particles". Figure 5.7 shows a filtering example for cellular automaton rule 110 — in Wolfram's numbering.

A filtering method similar to our proposed frequency-based coarse-graining — originally presented as a complexity metric for cellular automata — is introduced in (Wuensche 1999). The author proposes to progressively filter out cells in cellular automata's space-time diagrams according to read frequency of the rule table. Cells that originated from frequent rule table lookups are set to a quiescent or null state. The choice of threshold has to be decided by a user for each rule. Another notable difference is the method aims at making visualization of gliders easier without reducing the size of the grid or making more compact representations.

More recent work by (C. R. Shalizi, Haslinger, et al. 2006) uses the combination of a modified Lyapunov exponent approach with *statistical complexity* (C. R. Shalizi, K. L. Shalizi, and Haslinger 2004) to underline complex behaviors. However, the first method requires repeated perturbations and simulations of the system to study its sensitivity.

5.2.3 Scaling-up cellular automata

Hashlife (Gosper 1984) and other Game of Life-specific optimizations enable simulating a large number of cells for numerous timesteps. Nonetheless, these algorithms essentially exploit input redundancy. The regularity in patterns allowing such optimizations might indicate a lack of novel patterns being generated by the system.

This also means that Game of Life-based simulations are computationally reducible to a much simpler system, indicating that its computations are inefficient (Wolfram 2002). An optimally complex-behaving computational model should be impossible to predict except when computing its actual evolution step by step.

In the following, we used coarse-graining as a method for scaling down CAs in both time and space in order to make visualization of larger patterns and complex behaviors easier. The underlying fine-scale computations may be essential for these larger patterns to appear, hence the necessity to keep them. However, analogous to many natural processes (swarms, chemistry, cells in an organism, DNA), interesting behaviors might not be observable at the level of individual components — or small groups of components (individuals, single cells or molecules in the examples above). We view coarse-graining as a way to reduce a cellular automaton's space-time diagram to its most relevant parts while keeping primary dynamics in the background. The resulting diagram would ideally be an irreducible system.

5.3 Proposed coarse-graining of cellular automata

For reasons stated above, we introduce coarse-graining methods for cellular automata that are not reversible — information is discarded in the process. This process does not attempt to find shortcuts for the computations of a cellular automaton, but rather to select relevant parts of the space-time diagram and discard information irrelevant to the core behavior. For example, a standard glider in Game of Life spanning 3×3 cells could be replaced with a single cell moving diagonally when coarse-graining by a factor 3. This is because the actual oscillator's dynamics might not be relevant at this coarser scale.

Coarse-graining is akin to constructing *supercells* from blocks of individual cells. These supercells are assigned a new state and form a coarser partitioning of the initial grid which can be studied as its own system. In particular, complexity metrics or further coarse-graining can be applied to this new grid.

5.3.1 Frequency histogram coarse-graining

A simple coarse-graining is achieved by mapping blocks to a single *supercell* state according to the probability of this configuration appearing, given a previously constructed model. The easiest way to think of it is with a simple frequency counting model of the distribution of 2×2 blocks in a 2D CA. For a 2-state automaton, there are 16 possible supercell configurations. The simplest model for the occurrence of these blocks is their empirical frequency. Let us consider a CA with N blocks of 2×2 cells, let $S^{(in)} = \{\texttt{0000}, \texttt{0001}, \texttt{0010}, \ldots, \texttt{1111}\}$ be the set of 2×2 blocks and $s_i \in S^{(in)}$ be a given supercell. The probability p_i of observing supercell i on a grid G is estimated with

$$p_i = \frac{\text{count}_G(s_i)}{\sum_{s_j \in S^{(in)}} \text{count}_G(s_j)} \tag{5.1}$$

where $\text{count}_G(s_i)$ is the number of blocks matching (s_i) in G.

Supercells can then be assigned a particular state. We call the corresponding mapping $f : S^{(in)} \mapsto S^{(out)}$. $S^{(out)}$ can be chosen depending on the desired output or use. For instance, with $S^{(out)} = \{0, 1\}$ we can define f to map each supercell s_i as follows:

$$f(i) = \begin{cases} \texttt{0} & \text{if } \; p_i \geq \alpha \\ \texttt{1} & \text{if } \; p_i < \alpha \end{cases} \tag{5.2}$$

where α is a chosen threshold.

Partitioning the histogram

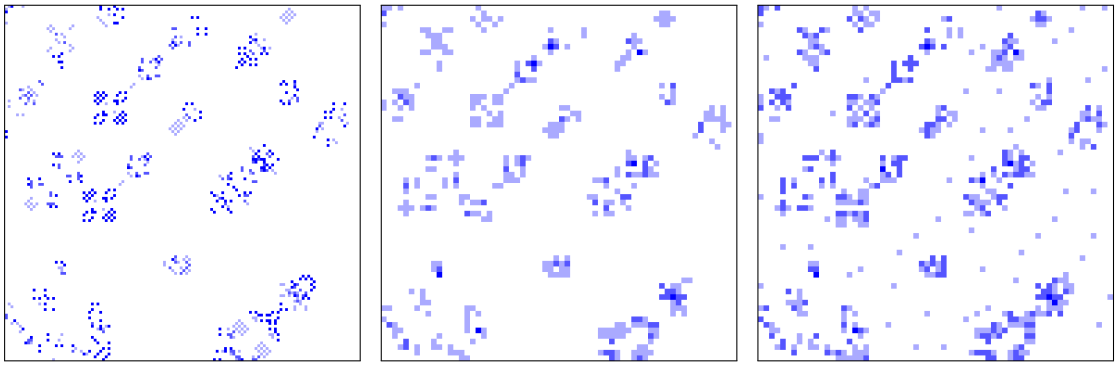
This method can be understood as partitioning the histogram of supercell frequency. In equation equation 5.2, supercells with low probability — with higher self-information — are mapped to state `1` whereas commonly occurring states are mapped to `0`.

Choosing a partition of the histogram is equivalent to selecting a suitable α — scalar for two output states, or vector $\alpha = (\alpha_1, \ldots, \alpha_n)$ for n output states. Therefore, one can map supercells to any number of target states (three or more) by partitioning the frequency histogram into any number of bins. Supercell distribution can be anything between uniform and very unbalanced, with a few supercells being overwhelmingly represented (background) and only a few occurrences of other configurations. The

chosen partitioning has to deal with both situations equally well. In the following, we use a uniform partitioning of the area under the negative log-histogram for elementary cellular automata — supercells are divided into two bins of equal summed negative logarithmic probability. For 2D CAs, we use the same method but with quadratic partition of the histogram ($1/k^2$ instead of $1/k$, with k the number of output states, chosen because of better visual results).

Dithering

Histogram partitioning introduces another set of parameters to be manually tuned, adding complexity to the procedure. An alternative way to produce an output image from the histogram is to use dithering. Dithering is an image processing technique commonly used to reduce large visual artifacts induced by quantization errors. Noise is added to the image during the quantization process to make the average local value of a set of pixels as close to their target continuous value as possible. The resulting image is created so as to match target continuous values with discrete values only — cell states in the grid. It can be seen as another way of partitioning the histogram with variable thresholds that depend on a running quantization error. Figure 5.2 shows a comparison of dithering and regular histogram partitioning (Floyd–Steinberg's algorithm was used (Floyd and Steinberg 1976)).

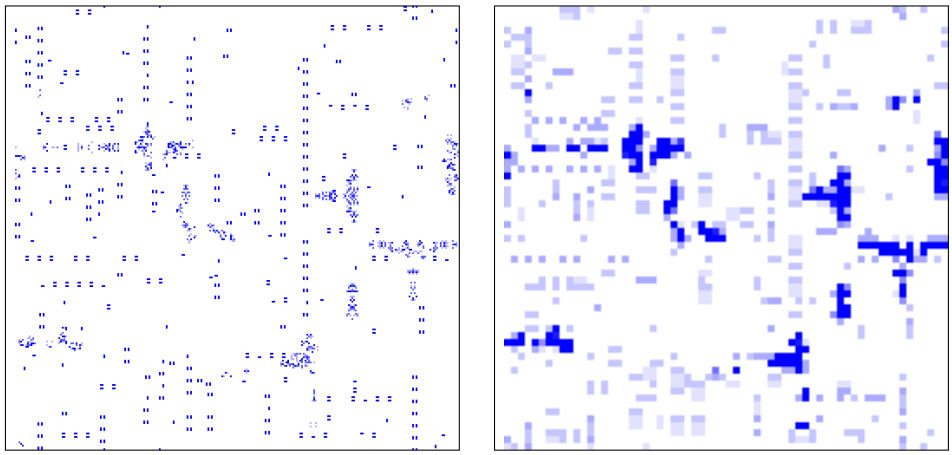


(a) Original CA **(b)** w/out dithering **(c)** with dithering

Figure 5.2 – Close-up view of coarse-graining effects on a 4-states CA rule (1 shade of blue per state). Both coarse-graining methods conserve many of the interesting structures. Dithering introduces additional artifacts on regular backgrounds. Fig. 5.2a shows actual states in the CA simulation on a 128×128 grid. Fig. 5.2b is a coarse-grained version of 5.2a with histogram coarse-graining, the grid is 64×64 cells. 5.2c is obtained with histogram coarse-graining and dithering (see Dithering).

Visualization

One advantage of this frequency histogram-based method is that it naturally highlights rarer events in the simulation grid, creating a "heatmap" of the simulation's activity. Since we sort supercells according to their observed frequency, the right choice of colors — e.g. progressively darker gradient — can lead to automatic highlighting of active regions of a cellular automaton. Figure 5.3 shows the same simulation both unprocessed and downscaled by a factor of 4 with coarse-graining. Although much coarser, Figure 5.3b is more readable than the base version, which is helpful when dealing with large grids*.



(a) Base grid **(b)** Coarse-grained

Figure 5.3 – Side-to-side comparison of a CA simulation and its coarse-grained version. The first simulation is 256×256 cells and the second has been coarse-grained to 64×64. Notice the interesting patterns on Figure 5.3a are hardly distinguishable. They are highlighted by histogram-based coarse-graining in Figure 5.3b.

Hierarchical coarse-graining

The above procedure can be applied recursively to the same cellular automaton or with larger block sizes to get a progressively coarser representation. Since information is systematically discarded in the process, it cannot be applied any number of times. For this reason, many 2D CAs exhibiting interesting behaviors at the micro-level but not at the macro-level have no remaining visible structure after reducing their scale several

*. Several figures in this chapter have animated versions, accessible at the project page of our paper (Cisneros, Sivic, and Mikolov 2020) https://hugocisneros.com/ALIFE-Paper-2020/

times with this method.

Because a simple model like frequency counting can be estimated quickly, hierarchical coarse-graining is easily applied to large grids, reducing the size by a factor of n (block size) every time. For instance, this property makes it suitable to search the cellular automata rule space for CAs behaving interestingly at multiple coarse-graining levels simultaneously.

5.3.2 Clustering

Another way to convert blocks of cells for coarse-graining is to distribute these blocks into a small number of clusters, where each group becomes the new coarse state.

Several distance functions may apply here, the most natural of which being Hamming distance, which measures how many states differ between two positions (Hamming 1950). It is defined for two strings of equal length n, $s_1 = [s_{(1,1)}, \ldots, s_{(1,n)}]$ and $s_2 = [s_{(2,1)}, \ldots, s_{(2,n)}]$, as the number of positions where the two strings differ:

$$\sum_{k=1}^{n} \mathbb{1}\left\{ s_1^{(k)} \neq s_2^{(k)} \right\}. \tag{5.3}$$

A supercell of $N \times N$ cells of a CA can be converted into a string to be compared to other blocks with the Hamming distance. For CAs, we limit ourselves to strings of digits representing states, i.e. $s_{(i,j)} \in \mathbb{N}$. We use a vanilla implementation of the K-means algorithm where clusters' centers are computed using a continuous average of position vectors rounded to nearest integer values. Clusters are initialized with randomly selected observations.

5.3.3 Autoencoders for coarse-graining

Instead of just relying on the amount of information of a given supercell's configuration, one can also try to automatically find a relevant representation with dimensionality reduction methods. Autoencoders are neural networks composed of an encoder part and a decoder part, originally designed to identify principal components of a collection of data points (Baldi and Hornik 1989; Hinton 1989; Kramer 1991). An encoder neural network converts data to a *latent* vector of smaller dimension than the original input.

Then, a decoder neural network reconstructs a vector with the same dimension as the input from this encoded *latent* representation.These models can automatically find an optimal constrained representation through minimizing a reconstruction loss between the original input and the reconstructed output.

We denote the encoder network with E and the decoder network with D. We frame the reconstruction problem as a N class classification problem with multiple components — one class per input state, one component for each cell of the K cells in a block. The reconstruction loss is the component-wise cross-entropy between the state of each input cell and the reconstructed state after $D \circ E$ is applied.

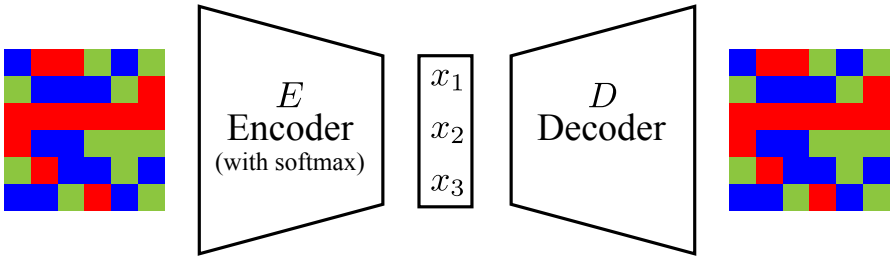


Figure 5.4 – Diagram of the autoencoder architecture used for coarse-graining. A block of 6×6 cells is encoded in a vector of fixed dimension. There are 3 components in the example. They can either represent a RGB color or a 3 states smaller automaton.

Figure 5.4 illustrates the autoencoder layout for coarse-graining. By adjusting the block size and dimension of the encoded vector, one can influence the amount of information conserved during encoding. Naturally, smaller blocks will be more easily represented in lower dimension.

The encoder has a softmax layer to ensure the coded state's components sum to one. Therefore, one can view this coded supercell as a mixture of states which can either be kept as is or converted to a discrete state by keeping the maximal component only. They are trained with stochastic gradient descent until convergence.

5.4 Results

We evaluate our proposed coarse-graining methods in the following two different ways:

— We compare our results on elementary cellular automata (ECA) to previous works

on particle and domain filtering.

— We use a metric which evaluates complexity of CAs introduced in (Cisneros, Sivic, and Mikolov 2019) in order to compare our methods' complexity metric scores of the coarse-grained systems and contrast the scores against a standard image processing baseline that computes local average of neighbouring cells followed by downscaling the grid. Using the complexity metric we measure to what extent the interesting behavior of cellular automata is conserved after coarse-graining compared to this image processing baseline.

In the following we begin by showing that a simple histogram-based coarse-graining is effective at detecting structures (such as gliders) in ECA space-time diagrams. Our method achieves results comparable with previous work, while being simpler to apply.

5.4.1 Domains and filtering

In the space-time diagrams of cellular automata, moving structures such as gliders are embedded in uniform or periodic backgrounds, or "domains". This domain is different depending on the rule: some ECAs have uniform backgrounds, checkerboard backgrounds or more complicated patterns (e.g. rule 110). (James P. Crutchfield and Hanson 1993) also identified chaotic domains, which cannot support regular gliders but have "walls" and "particles". Those correspond, respectively, to boundaries between two chaotic domains and propagating defects (localized structures with a pattern different from the domain) within a domain.

Our proposed coarse-graining methods offer interesting perspectives to filter cellular automata's space-time diagrams, which enables identifying gliders and studying the formation of large-scale patterns. We find that a simple histogram coarse-graining achieves results comparable to those reported in (Eloranta and Nummelin 1992; Hanson and James P. Crutchfield 1992, 1997; Wuensche 2011) for ECA rules 18 and 54. A similar approach was undertaken in (Wuensche 1999) in which the authors used the entropy of rule table lookup frequencies to filter out regular domains in the space-time diagrams of cellular automata and to identify gliders and domain boundaries. However, Wuensche's approach described in (Wuensche 1999) does not attempt to downscale space-time diagrams.

5.4.2 Results on elementary cellular automata

We apply frequency histogram-based coarse-graining on elementary cellular automata (ECA) and obtain space-time diagrams with suppressed background domains. Resulting partitions of ECAs' space-time diagram are similar to results reported by (Hanson and James P. Crutchfield 1992, 1997). Figure 5.5**(a)** and 5.6**(a)** show the space-time diagrams of rules 18 and 54 with random initialization. Boundaries between different background patterns in both Figures were obtained with coarse-graining; they are similar to boundaries obtained by Hanson and Crutchfield. We also observe the propagation of many of the same particles and defects without any prior information about the cellular automaton rule.

Particles in Rule 54 have been used to implement computations (N. Boccara, Nasser, and Roger 1991; Martinez, Adamatzky, and McIntosh 2014; Pivato 2007). Because the presented reduction reduces the size of the grid, it can merge some of those particles, sometimes resulting in ambiguities and gaps. However, our goal here is not to precisely describe particle interactions in order to manipulate or construct complex computations manually. Underlying computations described in the works above are still happening within our reduced CA simulation. We consider the *apparent* destruction of some of these fine-scale details acceptable in order to discover larger-scale complex behavior.

Our method is also arguably much simpler than computational mechanics (used by Hanson and Crutchfield) which requires some reverse-engineering of the rule and the construction of a finite-state transducer to generate output symbols. Although full automation has been demonstrated, this method introduces significant overhead (Rupe and James P. Crutchfield 2018). On the other hand, our method is sensitive to the quality of statistical estimation of the frequency histogram (see equation equation 5.1) and needs enough input examples to achieve a reasonable result — examples in Figure 5.5 and 5.6 used simulations with the width of 3000 cells, ran for 6000 timesteps to obtain reliable pattern frequency estimates.

In the Figures, we used the coarse-graining method introduced in Frequency histogram coarse-graining. Space-time diagrams are coarse-grained by a factor 2 to a binary automaton — each cell corresponds to a 2-cell block. These binary coarse-graining results in the Figures are labeled **(c)**. Because of the statistical nature of the domains of

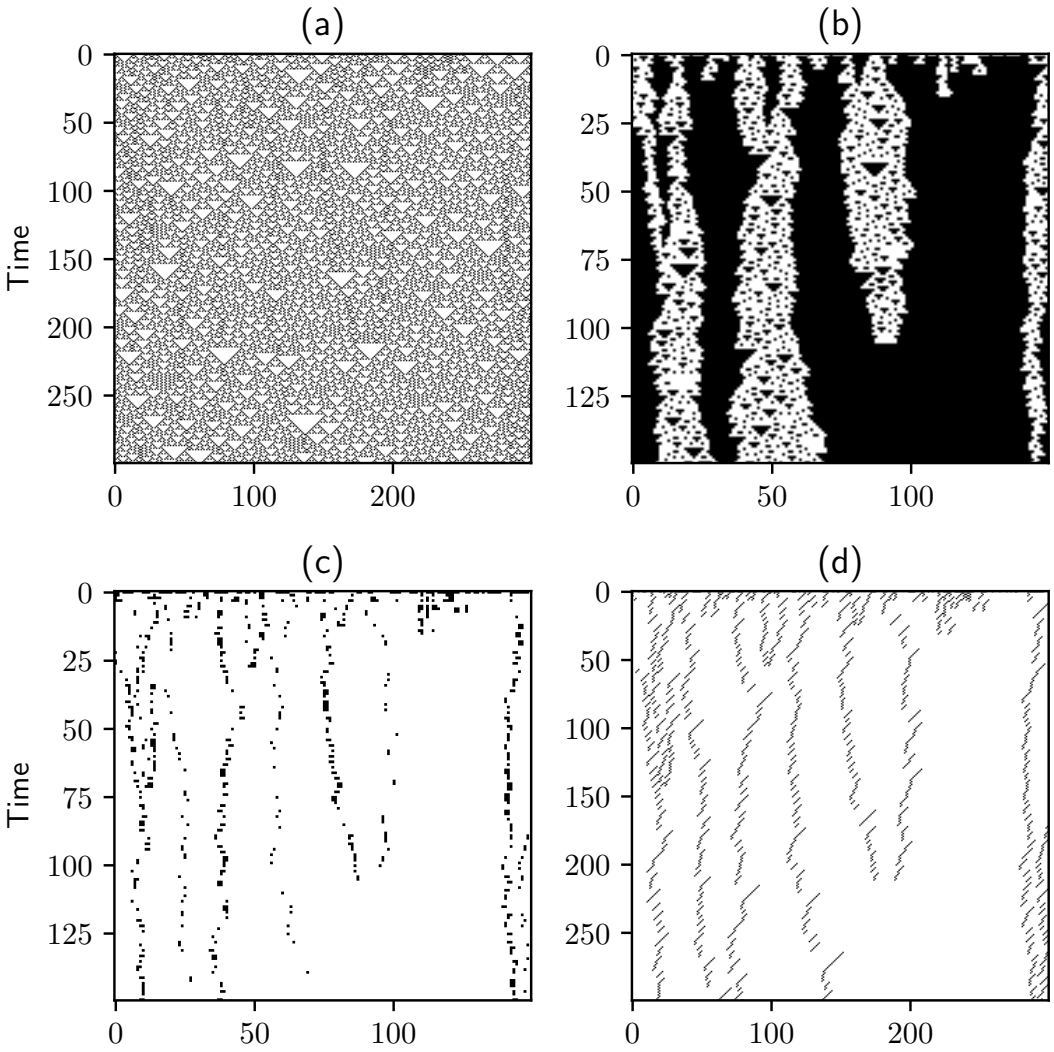


Figure 5.5 – Space-time diagrams for rule 18 in elementary cellular automata. **(a)** Standard rule 18 space-time diagram, starting from a random position. **(b)** Filtered domain with our frequency coarse-graining (even). **(c)** Our domain boundaries extracted from the filtered domains in **(b)**. **(d)** Domain boundaries computed according to (Hanson and James P. Crutchfield 1992). Note that **(a)** shows semi-chaotic behaviour, which is hard to interpret, whereas our method **(b)** highlights distinct domains within the disordered space-diagram in (a). The detected domains and domain boundaries from previous work (d) and ours (c) are very similar.

these 1D ECA's and the use of blocks of size 2, filtered domains differ depending on the starting position of coarse-graining. We distinguish an odd and even filtered domain.

Figure 5.5**(c)** is obtained by applying the element-wise `OR` operator to both the even and odd domain diagrams to merge them into a single space-time diagram. Figure 5.6**(c)** is obtained by computing differences between neighboring cells after the filtering process to highlight lines. Figure 5.7 is another example showing filtering of particles in rule 110.

5.4.3 Complexity metrics and coarse-graining

Coarse-graining is not only useful for detecting gliders and domains in space-time diagrams, but also as a tool to visualize large CAs. To evaluate the quality of our proposed coarse-graining methods, we compare complexity scores computed according to (Cisneros, Sivic, and Mikolov 2019) for different coarse-graining methods. This metric was shown to correlate well with a user study of interesting automata. It uses neural networks to estimate how easy it is to learn a compressed representation of a CA. We also computed the scores on downscaled CAs as a baseline. Local averaging is used for downscaling, with each block of N cells being replaced by its average value rounded to the nearest integer state.

Experiments begin by sampling 3600 cellular automata rules with 3 or 4 states. We apply the complexity metric on a randomly initialized simulation on a 512×512 grid of cells. The top 100 rules with the highest complexity scores, which should correspond to rules with interesting behaviors, are then used for coarse-graining. We apply coarse-graining on grids of 4096×4096 cells, scaling the grid down by a factor of 8, and compute the complexity metric also on the reduced grid. Figures reported in Table 5.1 are percentages of rules still considered interesting (above the selection threshold for the first step of the process) after coarse-graining. The higher this number is, the more a method is able to conserve complex and interesting behaviors after the reduction.

Results in Table 5.1 suggest that using our proposed methods seems largely beneficial for studying complexity in large systems. Histogram and autoencoder methods are superior to downscaling using k-means and local averaging. This could be attributed to the fact that contrary to the latter two, the histogram and autoencoder both represent

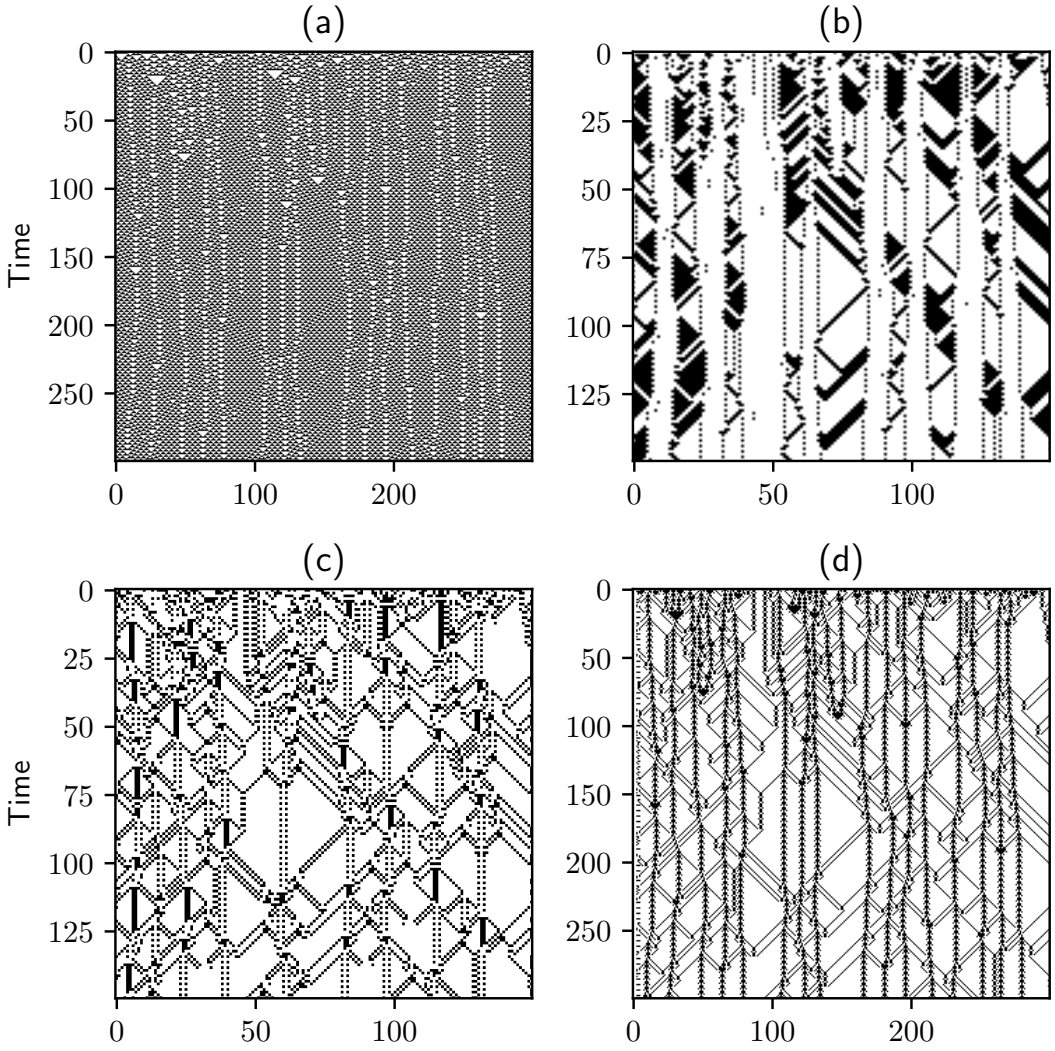


Figure 5.6 – Space-time diagrams for rule 54. **(a)** Space-time diagram of standard rule 54, starting from a random position. **(b)** Filtered domain with our frequency coarse-graining (even). **(c)** Particles filtered from the domains in **(b) using our method**. **(d)** Domain boundaries computed using computational mechanics (Hanson and James P. Crutchfield 1997). Please note that particles are detected equally well using computational mechanics (d) and our (simpler) frequency-based method (c). Some close-by particle trails are merged using our method.

Local-averaging baseline	K-Means	Histogram	Autoencoder
19.3%	40.4%	82.4%	84.2%

Table 5.1 – Experimental results — Percentage of rules classified as interesting after reduction with our 3 proposed methods (K-means, Histogram, Autoencoder), compared to a local averaging baseline.

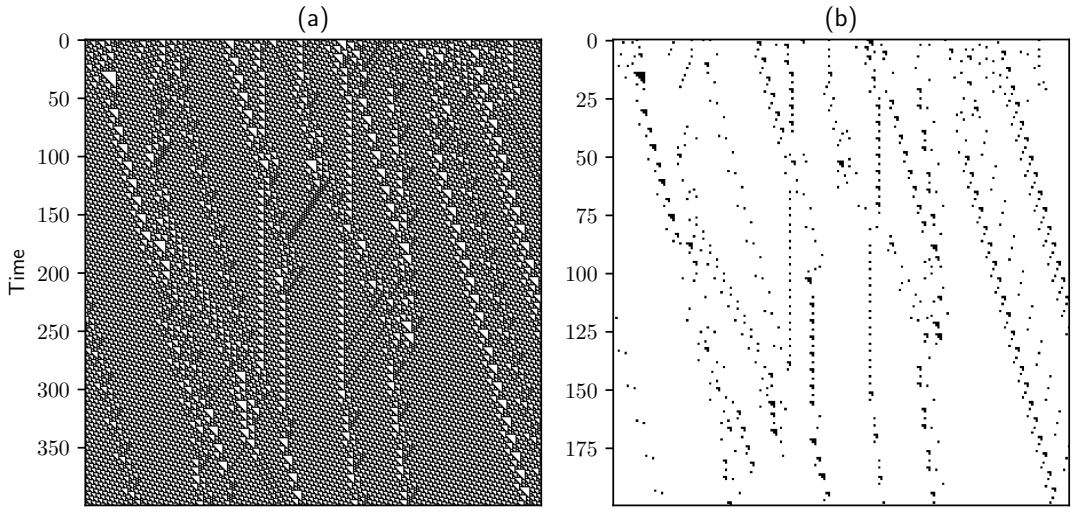


Figure 5.7 – Spate-time diagram of rule 110 (**a**) and filtered particles using our histogram-based coarse-graining (**b**). Structures propagating in time (vertical axis) and space (horizontal axis) become clearly visible in **(b)** as vertical and diagonal lines.

well anomalies (rare events). This is because rare events are explicitly captured and kept by the histogram method. They also represent useful information that may be kept for reconstruction using the autoencoder.

5.4.4 Discussion

Downscaling by local averaging is not an effective solution to the coarse-graining problem for several reasons. In particular, it tends to favor the majority state in a supercell because of the averaging effect. Thin structures spanning only a few cells placed on a uniform background are likely to disappear after coarse-graining, although they may still be relevant with respect to large-scale patterns. The histogram-based method explicitly encodes those rare events in a supercell, even if their size is relatively small compared to the supercell size.

Figure 5.8 is a qualitative comparison of coarse-graining methods. This cellular automaton was selected from the experimental dataset. When simulated on large grids, it generates large linear structures that are 4-cells wide. These structures disappear after downscaling by averaging because the background dominates the average. Other methods correctly highlight these structures when downscaling the grids by a factor of 8. In Figure 5.9, we show another rule that was selected for its high complexity score

on multiple coarse-graining scales from our dataset. The CA has significantly different dynamics depending on the chosen scale. Example 5.9a is a spontaneously occurring stable oscillating glider with period 3. Large structures emerge from these simple gliders when observing large grids. The online project page [†] shows animated examples for Figure 5.9, emphasizing the advantage of using coarse-graining for visualization. Figures 5.10 and 5.11 show more comparisons of large-scale coarse-graining of interesting CAs.

A crucial advantage of the frequency histogram method is its speed and ease of implementation compared to autoencoders. Other than a few hyper-parameters for partitioning the histogram, no training or tuning is needed to produce the coarse-grained output.

5.5 Conclusion

We intend to use these coarse-graining methods to find cellular automata (CA) which exhibit interesting behaviors at multiple scales. Figure 5.9 shows an example of such a CA. We observe various dynamics depending on the scale, from simple oscillating gliders to large wave-like patterns composed of thousands of gliders. It demonstrates that observing multi-scale behaviors within those automata is possible. The existence of 2D cellular automata with disordered behaviors at the smallest level but organized at coarser scales, similar to hidden patterns in rule 18, would also be of great interest.

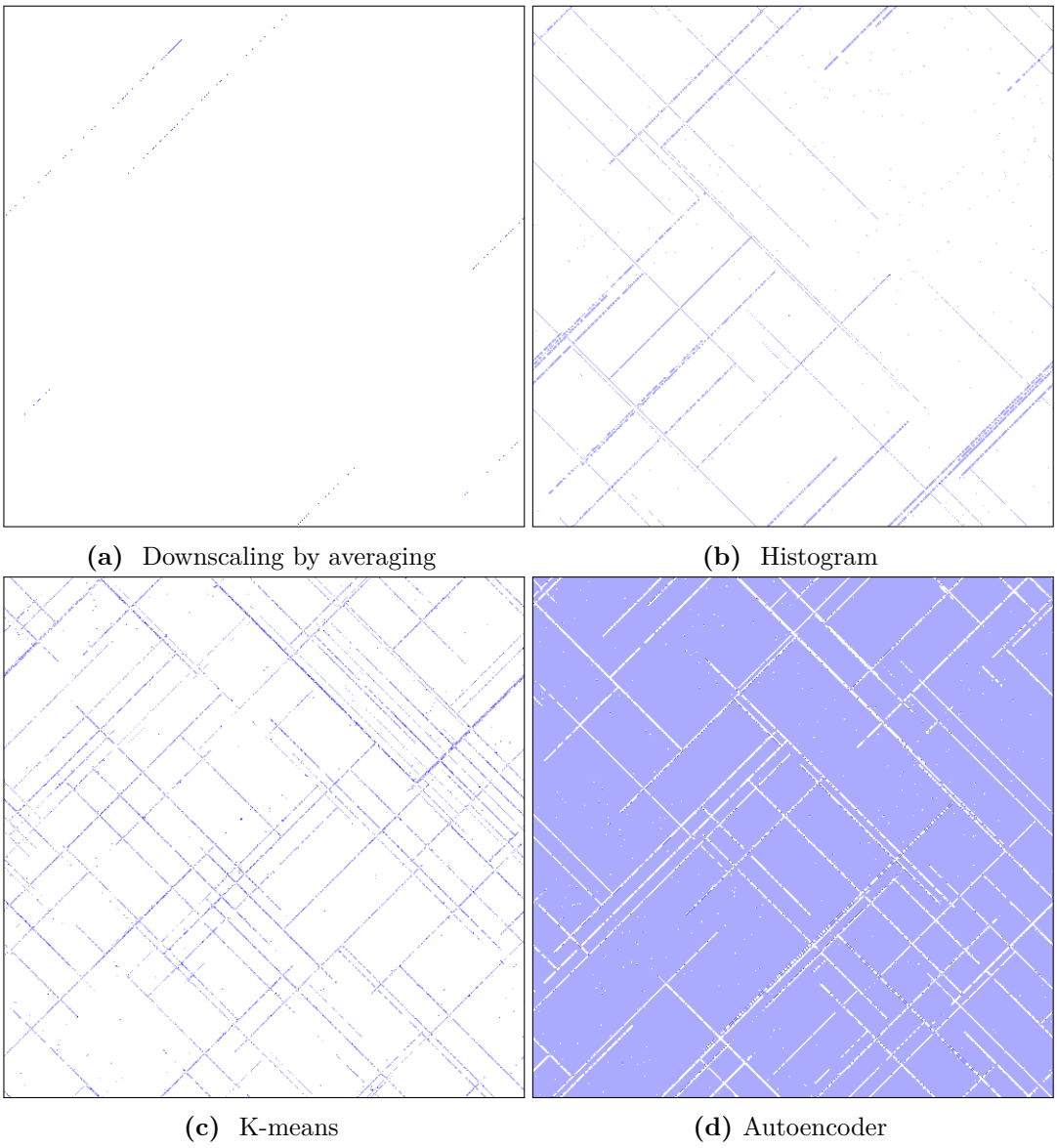
Cellular automata are powerful computational models. Some of them have been shown to be Turing-complete, and can thus be expected to support arbitrarily complex computations (Berlekamp, Conway, and Guy 2001; Cook 2004). Naturally, most interesting CAs spontaneously generate a fraction of available computations at a time, usually supporting a few stable oscillators or moving structures. Proofs of universality for these CAs required careful design of computational devices out of these stable oscillators and structures, resulting in very brittle and inefficient universal computers. In practise, only elementary functions — such as density classification, binary addition, etc. — can be implemented. This requires searching for CA rules specifically targeted at a particular function (Melanie Mitchell, James P Crutchfield, and Das 1996; Sapin, Bailleux, and Jean-Jacques 2003; Wolfram 2002). Hierarchies are central to naturally occurring com-

†. https://hugocisneros.com/ALIFE-Paper-2020/

plex phenomena (Simon 1962), and may be required for robust and complex processes to emerge in CAs.

Viewing space-time diagrams of cellular automata is akin to visualizing a foreign computer design. Cellular automata are manipulating information, registers and instructions in parallel in the form of cell states. We believe visualization tools proposed in this chapter can help understand computations in those unconventional computers. By reducing available information to its essential parts, we attempt to distill the content of the space-time diagram with as little prior information as possible. Future work could focus on identifying some known simple computational primitives within cellular automata and understanding how our visualization can help to find them.

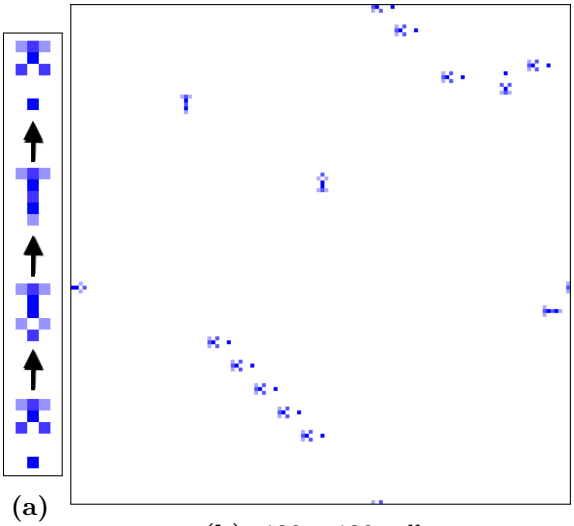
These methods also enable apprehending large grid sizes for which even image processing algorithms begin to show limitations. Complexity metrics and CA classification techniques can be extended to these reduced large grids and could lead to the discovery of CAs with — similar to life and physical processes — significantly different dynamics at multiple scales that could in turn be a basis for artificial evolution.



(a) Downscaling by averaging **(b)** Histogram

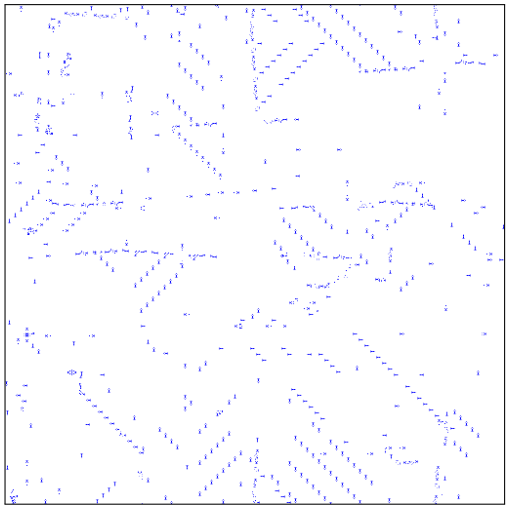
(c) K-means **(d)** Autoencoder

Figure 5.8 – Qualitative comparison of coarse-graining methods Simulations are on grids of 4096×4096 cells coarse-grained to 512×512. Most lines are barely visible with downscaling 5.8a, but are visible in 5.8b-5.8d. Coarse-graining helps visualize linear structures that would be hard to see otherwise.

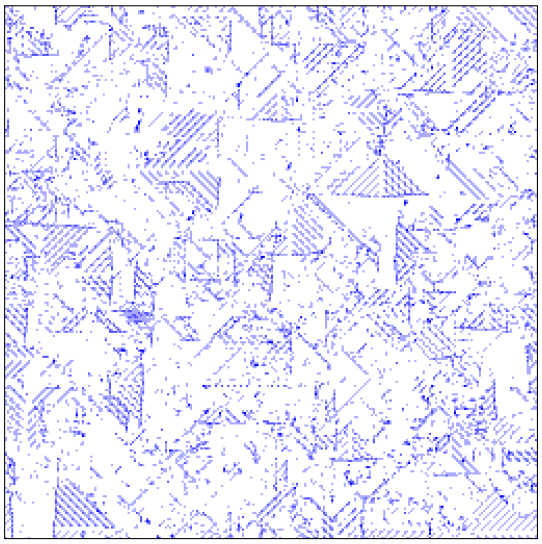


(a)

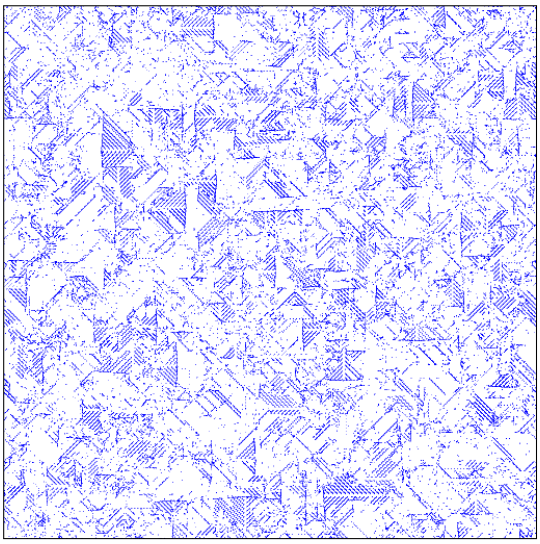
(b) 128 × 128 cells



(c) 512 × 512 cells

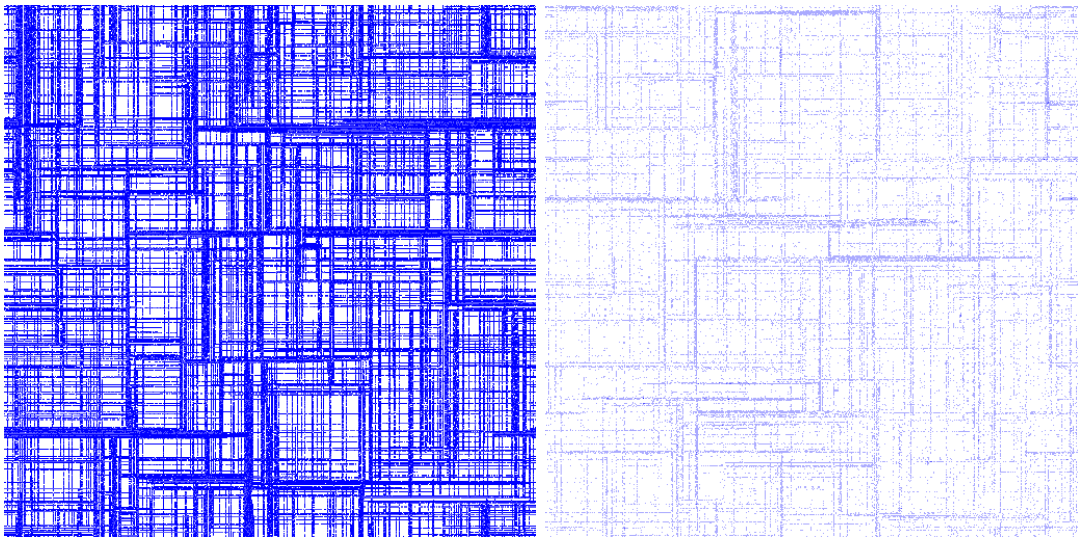


(d) 2048 × 2048 cells coarse-grained to 256 × 256.



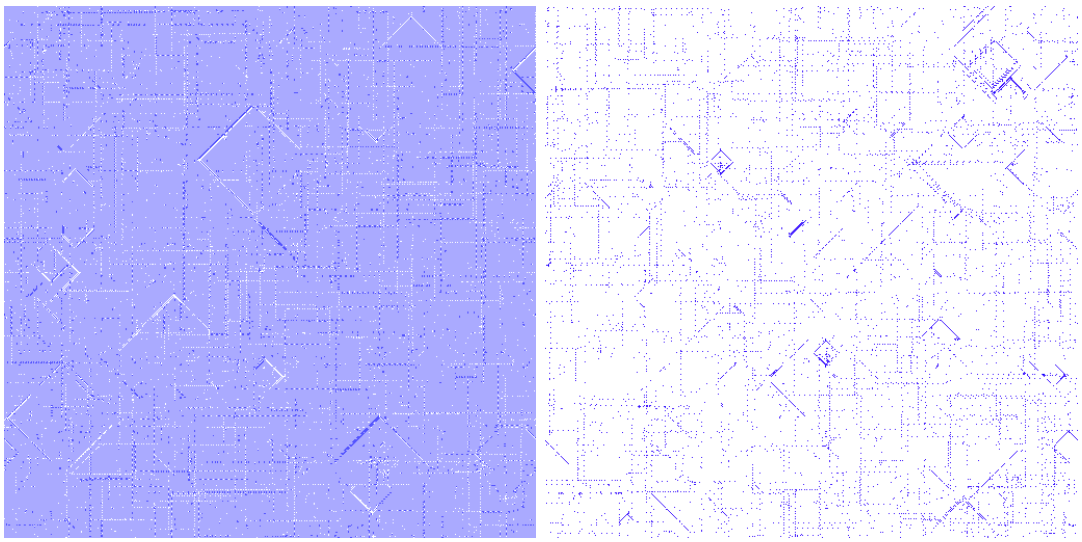
(e) 4096 × 4096 cells coarse-grained to 256 × 256.

Figure 5.9 – Changing CA dynamics at multiple scales. (a) shows a single glider oscillating between 3 positions. Such gliders emerge spontaneously from a random initialization of a small grid, as shown in (b). When scaling the grid up, trails of gliders begin to appear, creating moving straight and diagonal lines, as shown in (c). Scaling up even more, individual gliders are not visible anymore, as shown in (d). In an even larger grid, shown in (e), many more triangular-shaped waves travel and collide with each other. Please note that (d) and (e) are coarse-grained to 256 × 256. Otherwise, the patterns would not be visible.



(a) Autoencoder **(b)** Histogram

Figure 5.10 – A CA with 4096×4096 cells coarse-grained to 256×256. Comparison of the histogram (5.10b) and autoencoder (5.10a) methods.



(a) Autoencoder **(b)** Histogram

Figure 5.11 – A CA with 4096×4096 cells coarse-grained to 256×256. Comparison of the histogram (5.11b) and autoencoder (5.11a) methods.

Chapter

6 Learning efficiency in deep reservoir computing

It is common to evaluate the performance of a machine learning model by measuring its predictive power on a test dataset. This approach favors complicated models that can smoothly fit complex functions and generalize well from training data points. Although essential components of intelligence, speed, and data efficiency of this learning process are rarely reported or compared between different candidate models. In this chapter, we introduce a benchmark of increasingly difficult tasks together with a data efficiency metric to measure how quickly machine learning models learn from training data. We also highlight that within complex systems, there are computations taking place that can be harnessed for various purposes. To that end, we compare the learning speed of some established sequential supervised models, such as RNNs, LSTMs, or Transformers, with relatively less-known alternative models based on reservoir computing, which is a tool for harvesting the computations happening within a complex system. The proposed tasks require a wide range of computational primitives, such as memory or the ability to compute Boolean functions, to be effectively solved. Surprisingly, we observe that reservoir computing systems that rely on dynamically evolving feature maps learn faster than fully supervised methods trained with stochastic gradient optimization while achieving comparable accuracy scores. The code, benchmark, trained models, and results to reproduce our experiments described in this chapter are available at https://github.com/hugcis/benchmark_learning_efficiency/.

6.1 Introduction

Most machine learning models are evaluated by measuring performance on a specific dataset or task. Learning efficiency – the ability to learn, generalize, and adapt quickly from a few examples – is crucial for practical intelligence (Kanazawa 2004) and low-data machine learning applications, but rarely used to evaluate models. Supervised learning systems are theoretically limited in their learning speed by the optimization algorithms used for training. These algorithms, such as stochastic gradient descent (SGD), have various speed guarantees depending on the structure of the function to be optimized (Bottou, Curtis, and Nocedal 2018). However, when intelligent beings learn, they appear to quickly reuse past knowledge and progressively improve over time. Their learning speed depends on a dynamically evolving internal state. To measure the learning efficiency of various systems, in this work we propose the Weighted Average Data Efficiency (WADE) metric based on the time taken to reach several test accuracy checkpoints. We also design a simple modular benchmark composed of a set of sequential tasks. They begin with the task of recognizing a simple periodic sequence in an input string and end with elaborate question-answering tasks that require counting occurrences of patterns and long-term memory.

Established sequential supervised models such as recurrent neural networks (RNNs; Elman 1990), long short-term memory networks (LSTMs; Hochreiter and Schmidhuber 1997) or Transformers (Vaswani et al. 2017) lack essential properties such as learning beyond the training phase or the ability to adapt over time after being trained. These models can also be expensive to train, requiring a large number of labeled training examples to reach reasonable performance, leading to poor learning speeds. In this work, we use the newly proposed WADE metric and the benchmark dataset to experimentally compare the learning speed of these well-established models, such as RNNs, LSTMs, and Transformers, with less explored *reservoir computing models*.

Reservoir computing is a computational framework that aims to exploit the states of a complex dynamical system. The simplest example of a reservoir computer is a RNN with frozen weights. This special RNN performs random manipulation on its hidden state in reaction to each new input. Interestingly, it has been shown that with a specific initialization of the frozen weights, these RNNs (called Echo state networks)

can keep a memory of past inputs (Jaeger 2001). Usually, a standard linear regression is added as a decoder to extract valuable representation from the hidden state for some downstream task. Freezing the weights of these recurrent models is useful when available supervision is very limited or non-existent or for reinforcement learning with sparse rewards since direct training would be impossible. In such cases, a reservoir computer creates a continuously evolving pool of random functions that can be combined using the last trainable layer.

When evolving in response to input stimuli, complex recurrent systems such as RNNs are building dynamically changing representations of data within their internal state (Nino Boccara 2010). We know that these internal states can be interesting on their own because of their ability to self-organize and exhibit increasingly complex behaviors (Allen and Strathern 2003; C. H. Bennett 1995; Cisneros, Sivic, and Mikolov 2019; Goldstein 2011; Koppel and Atlan 1991). We wish to investigate whether complex dynamical systems — in particular RNNs with frozen weights (echo-state networks) (Jaeger 2001) and reservoir cellular automata (Yilmaz 2014) — create representations that allow them to learn faster, as measured by our metric.

Contributions. In this chapter, we make the following main contributions: First, we introduce the Weighted Average Data Efficiency (WADE) metric to measure the learning speed of various learning systems and use it to benchmark a few standard models on the IMDB text classification task (Maas et al. 2011). Second, we present a benchmark of language-based tasks of increasing difficulty to evaluate the learning speed in different conditions. The proposed tasks require a wide range of computational primitives, such as memory or the ability to compute Boolean functions, to be effectively solved. Third, we study the learning speed of reservoir computing learning models and compare them with more standard supervised solutions.

6.2 Related work

The WADE metric is a generalization of the *Time-to-threshold* metric (M. E. Taylor and Stone 2007; M. E. Taylor, Stone, and Liu 2007) introduced for measuring transfer learning in reinforcement learning contexts. In general, the Time-to-threshold is simply

defined as the number of training steps needed to reach a fixed threshold performance. However, this definition leaves open the choice of threshold or the definition of a training step. WADE alleviates this issue by aggregating several of these thresholds into a single number that summarizes the learning speed.

Other metrics for measuring how quickly a model adapts to new tasks have been introduced in the context of transfer learning, few-shot and zero-shot learning. In few-shot learning, one tries to obtain the best performance for a particular task using a small amount of labeled data compared to the task's fully supervised equivalent (Y. Wang et al. 2020). This correlates with a model's learning speed, but these problems often measure how much prior information about similar data has been encoded in the models. With our benchmark and the WADE metric, we explicitly measure the number of steps to reach multiple test accuracy values using all the data needed, effectively emphasizing data efficiency.

Sample efficiency has also been studied in the context of reinforcement learning. (Chevalier-Boisvert et al. 2018) use the number of demonstrations before a task is solved to measure sample efficiency. This requires defining what solving the task means, which may vary from task to task. Another approach is to measure performance (cumulated reward, accuracy, etc.) after a fixed budget of training steps (Yarats et al. 2019). In this case, the most efficient model is the one that achieves the best performance within the allocated budget. In other cases, the sample efficiency is mentioned but not explicitly measured and one has to examine the learning curves (Buckman et al. 2018). The WADE metric is a general approach to measure the learning efficiency of machine learning models. We use it to benchmark a few standard models on the IMDB text classification tasks (Maas et al. 2011) and propose a set of modular and extensible language-based tasks.

Synthetic tasks such as ours have played a vital role in a series of crucial advances in machine learning algorithms. For example, the XOR problem has partially motivated the development of neural networks (Minsky and Papert 1972; Rumelhart, Hinton, and Williams 1985), and the *circle and ring* dataset has inspired the creation of novel clustering algorithms (Ng, Jordan, and Y. Weiss 2001). The design of synthetic datasets has also been an essential component of the development of learning algorithms with memory and general computational capabilities (Graves, Wayne, and Danihelka 2014;

Hochreiter and Schmidhuber 1997; Joulin and Mikolov 2015; Richardson et al. 2020; Weston et al. 2016).

Other tasks are based on real datasets with artificial manipulations (Goodfellow et al. 2014; Krizhevsky and Hinton 2009; Nguyen et al. 2017; Srivastava et al. 2013). The goal of our dataset is to be truly progressive in difficulty yet simple to understand and extend, to allow applications in the field of online learning, and to easily understand a model's basic computational capacities. Combined with our metric, it enables us to measure learning speed across a range of conditions. In contrast to similar synthetic datasets, we built this benchmark so that the last task is vastly more complicated than the first and could still be extended to more complex examples.

6.3 A benchmark for reservoir computing

To measure the learning speed of candidate systems and their ability to improve over time, we propose a performance metric and a standardized set of tasks. We want to select those systems that quickly and reliably adapt and learn from new inputs. For this purpose, we introduce the Weighted Average Data Efficiency (WADE) metric. It aggregates the speed at which a model reaches several test accuracy checkpoints. We describe the metric in more detail in Section 6.3.1.

To reliably compare learning speeds for various systems on a shared foundation, we also introduce a novel dataset described in Table 6.1. It is made up of sequential tasks that begin with straightforward pattern recognition and progressively increase in complexity to approach the complexity of natural language and other complex real-world tasks.

We do not focus on the prediction performance of our models but rather on their data efficiency — the number of example sequences they need to learn from before reaching a target accuracy on a validation set.

Standard benchmarks and metrics such as those introduced in this chapter have always been essential in advancing various aspects of machine learning. For example, the LSTM network demonstrated a superior memory capacity on a set of synthetic tasks designed to challenge the memory of sequential learning systems (Hochreiter and Schmidhuber 1997). Our goal with this benchmark is to emphasize measuring learning

Task id	Name	Description
1	Simple periodic pattern identification	Identify a simple periodic pattern.
2	Harder periodic pattern identification	Identify a periodic pattern with an arithmetically increasing period.
3	Symbol counting	Count symbols from a sequence.
4	Pattern counting	Count patterns (delimited group of symbols) from a sequence.
5	Simple question answering	Answer simple YES/NO questions from a single prompt.
6	Harder question answering	Answer simple YES/NO questions from a single prompt with a more extensive vocabulary.
7	Question answering with world definition	Answer YES/NO questions from a sequence of prompts.
8	Question answering with world definition and counting	Answer YES/NO and counting questions from a sequence of prompts.
9	Adjective question answering	Answer YES/NO and adjective questions from a sequence of prompts.
10	Adjective question answering and counting	Answer YES/NO, adjective, and counting questions from a sequence of prompts.

Table 6.1 – General description of all the tasks in the benchmark.

speed across tasks of varying difficulties with a range of computational requirements rather than focusing on performance only. We describe the performance metric and the benchmark next.

6.3.1 Performance metric

We introduce the Weighted Average Data Efficiency (WADE) metric as a way to measure how quickly a model learns using a weighted average of inverse times taken to reach various test accuracy *checkpoints* over time.

It is computed for an evenly distributed set of target accuracies $\mathbb{A}$. They represent the *checkpoints* at which the speed of learning is estimated. For example, we may choose $\mathbb{A} = [0.1, 0.2, 0.3, 0.4, \ldots, 1.]$. The metric is then calculated as

$$\mathrm{WADE}(\boldsymbol{a}) = \frac{1}{\sum \alpha} \sum_{\alpha \in \mathbb{A}} \frac{\alpha}{\mathrm{T}(\alpha, \boldsymbol{a})}, \tag{6.1}$$

where $\boldsymbol{a} = (a_0, a_1, \ldots, a_n)$ is a sequence of test accuracies achieved by the evaluated system sampled at different training steps. The quantity a_i typically corresponds to the accuracy reached after seeing i examples, and $\mathrm{T}(\alpha, \boldsymbol{a})$ is the number of steps in the sequence $\boldsymbol{a}$ needed to reach an accuracy of α. It is defined as

$$\mathrm{T}(\alpha, \boldsymbol{a}) = \min \{ i \in \{1, \ldots, n, +\infty\} \mid a_i \geq \alpha \}. \tag{6.2}$$

We also define $\mathrm{T}(\alpha, \boldsymbol{a}) = +\infty$ if the accuracy value α is never reached in $\boldsymbol{a}$. This is equivalent to appending an additional term $a_{+\infty}$ to $\boldsymbol{a}$, always set to the maximum accuracy 1. Note that by construction, $\mathrm{T}(\alpha, \boldsymbol{a})$ is in $[1, +\infty[$.

Since T can be $+\infty$ we define $\frac{1}{+\infty} = 0$ for the quantity in equation 6.1 to always exist. A visual intuition of $\mathrm{T}(\cdot, \cdot)$ is given in figure 6.1.

The choice of checkpoints $\mathbb{A}$ does not need to be tuned in any specific way because $\mathrm{WADE}(\boldsymbol{a})$ quickly converges to a single value when $\mathbb{A}$ approaches the continuous interval $[0, 1]$. The approximation is good enough as long as $\mathbb{A}$ is not too coarse (more than ten elements was enough in our experiments), and the WADE values computed from the same set $\mathbb{A}$ are comparable.

The time-to-threshold T is always greater than or equal to 1 step for any threshold

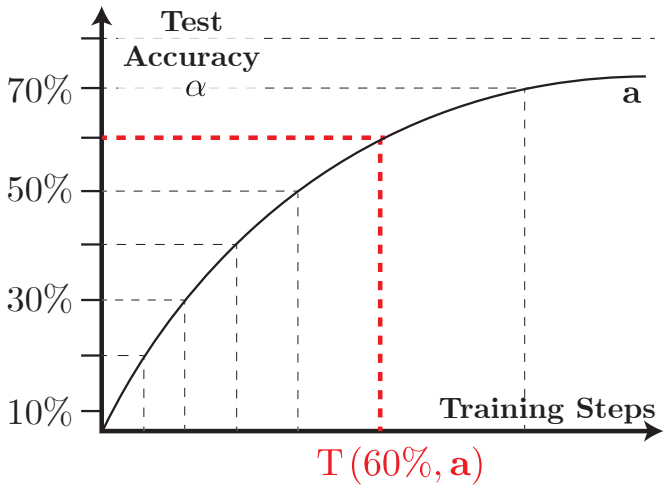


Figure 6.1 – Illustration of the calculation of $\mathrm{T}(\cdot,\cdot)$, representing the number of training steps (x axis) needed to reach a certain test accuracy α (y axis) from a learning curve. In this example, $\mathbb{A} = [0.1, \ldots, 0.8]$ (y axis). $\mathrm{T}(0.6, \boldsymbol{a})$ is highlighted in red. $\mathrm{T}(0.8, \boldsymbol{a}) = +\infty$ as the accuracy of 0.8 is never reached.

and sequence of accuracy scores. We have $\forall\, \alpha \in [0,1]$, $\forall\, \boldsymbol{a} = (a_n)_{I \subset \{\mathbb{N} \cup \{+\infty\}\}}$, and we have

$$\frac{1}{\mathrm{T}(\alpha, \boldsymbol{a})} \leq 1, \tag{6.3}$$

and therefore we always get $0 \leq \mathrm{WADE}(\boldsymbol{a}) \leq 1$. The metric is equal to 0 for systems that never get past the smallest possible accuracy, while 1 corresponds to reaching a perfect test accuracy in one single training step. Such a system would also be considered to be performing well according to the underlying performance metric with which it is usually evaluated. Therefore, maximizing WADE also maximizes performance.

6.3.2 Description of tasks in the benchmark

This section provides a more detailed description of each task in our benchmark. The tasks are designed to be language modeling tasks, where the goal is to predict some tokens from sequences of previously processed tokens. An overview of the tasks is given in Table 6.1. The tasks are divided into three groups: (i) binary tasks with only binary symbols, (ii) general symbolic tasks – symbolic manipulations with arbitrary symbols, and (iii) language-based tasks, where symbols represent words in English and behave like a language. We introduce this benchmark together with the WADE metric, but both can be used in other contexts as well to measure the learning speed of other systems. Individual sentences are generated and divided into a training set and a test set for

periodic evaluation of the test accuracy. We give a more detailed description of each task below*:

Binary

Simple periodic pattern identification. The goal of the periodic binary task is to teach the model a fixed-length regular pattern. As the system is presented with new binary input tokens, it has to learn the periodic pattern on the fly and correctly predict the next token. A pattern of size n is chosen at random and repeated k times to produce a sequence of length $n \times k$. Examples include:

`01010101010101010101010101010101`	Pattern with period 2
`00110011001100110011001100110011`	Pattern with period 4
`01101101101101101101101101101101`	Pattern with period 3

Harder periodic pattern identification. For this task, we also draw a random binary pattern of size n. Each of its symbols is repeated k times, with k increasing monotonically from 1. A successful model must learn the pattern on the fly and correctly implement the arithmetic increase in the size of the period. We set the pattern length to increase by 1 every period in our experiments, but this value can be changed.

Symbolic counting

These tasks consist of reading patterns from an input sequence and answering a simple query about the number of patterns. Unlike previous tasks, these require implementing a form of addressable memory that can be queried after the prompt has ended.

Basic symbol counting. The first version of the counting task focuses on counting single symbols from an input sequence. The sequence ends with a *query* for the count of one of the symbols. The goal is to predict the last token (in bold) of sequences of the following form:

∗. The tasks are also available as a Python package on GitHub.

$$\underbrace{\texttt{AABBCBABAAB}}_{\text{Input symbols}}\ \underbrace{\texttt{x}}_{\text{QS}}\ \texttt{A}\ \underbrace{\texttt{5}}_{\text{Answer}}$$

The symbol `x` is the query symbol (QS) that marks the beginning of the query. In the first example above, the goal is to predict the token `5` because the symbol `A` appears 5 times. As detailed in Sect. 6.6, we represent these nonbinary symbols with one hot encoding, so the numerical nature of some tokens is not encoded a priori.

Pattern counting. This aim of this task is to count the number of occurrences of delimited patterns instead of single symbols. A sequence is still divided between a prompt — before `x` — and a query — after `x`. One has to predict the symbol coming after each separator symbol (S) `y` in the query part of the sentence. For example, sentences are of the form:

$$\underbrace{\texttt{AA}}_{\text{Pattern 1}}\ \underbrace{\texttt{y}}_{\text{S}}\ \underbrace{\texttt{BBC}}_{\text{Pattern 2}}\ \underbrace{\texttt{y}}_{\text{S}}\ \underbrace{\texttt{BAB}}_{\text{Pattern 3}}\ \underbrace{\texttt{y}}_{\text{S}}\ \underbrace{\texttt{AA}}_{\text{Pattern 4}}\ \underbrace{\texttt{y}}_{\text{S}}\ \underbrace{\texttt{B}}_{\text{Pattern 5}}\ \underbrace{\texttt{x}}_{\text{QS}}\ \underbrace{\texttt{AAy}}_{\text{Query 1}}\ \underbrace{\texttt{2}}_{\text{Answer 1}}\ \underbrace{\texttt{By}}_{\text{Query 2}}\ \underbrace{\texttt{1}}_{\text{Answer 2}}$$

Multiple queries are presented successively, which requires keeping and being able to retrieve several counts simultaneously. A query is composed of a pattern, a separator symbol, and the pattern count that the system should predict.

Basic language understanding

To make the tasks progressively more complex, we steer them towards general language understanding tasks. The tasks described below are generated automatically, but gradually incorporate more complex skills required for advanced language processing. The last task is a step towards understanding the general language albeit with a limited vocabulary.

Elementary question answering (QA). This task introduces elements of natural language. Each example is composed of a stated fact and a question about that fact. A sentence is constructed from a few basic elements: (i) Names (e.g., `JOHN`, `JAMES`, etc.), (ii) Verbs (e.g., `HEAR`, `SEE`, etc.), (iii) Answers (`YES` or `NO`), (iv) Additional words and symbols (`I, DO, NOT, AND, BUT, ?, .`).

A random subset of names is selected, and we generate a random prompt/question pair from it. The question is drawn to ensure an equal proportion of positive and negative answers. For example, sentences may look like this:

```
I HEAR JOHN AND PAUL . DO I HEAR PAUL ? YES
I SEE JOHN BUT I DO NOT SEE PAUL AND TOM . DO I SEE TOM ? NO
```

The only token to predict is the binary answer `YES` or `NO`.

Question answering (QA) with adjectives. This task extends the previous task by adding adjectives and modifiers to the object names. The queries may be about the subject-verb relation or the subject-adjective relation.

```
I SEE A SMALL BANANA . WHAT IS THE SIZE OF THE BANANA I SEE ? SMALL
I SEE A LARGE GREEN APPLE BUT I DO NOT SEE A RED APPLE .
    DO I SEE A LARGE APPLE ? YES
I SEE A SMALL GREEN APPLE BUT I DO NOT SEE A BANANA .
    WHAT IS THE COLOR OF THE APPLE I SEE ? GREEN
```

Here the task output space is slightly larger because the model may be predicting `YES`, `NO`, `SMALL`, `GREEN`, etc.

Question answering (QA) with world definition. This task introduces more complex configurations in which the state of the world is defined in one or more sentences, and an unknown number of questions follow. This is more akin to a real-world conver-

sation where stored facts should be remembered longer and accessed on demand. For example, below we show a generated group of sentences followed by several questions:

```
I SEE A SMALL BANANA .
I SEE A LARGE GREEN APPLE BUT I DO NOT SEE A RED APPLE .
I SEE A SMALL GREEN APPLE BUT I DO NOT SMELL A BANANA .
    WHAT IS THE COLOR OF THE APPLE I SEE ? GREEN
    HOW MANY THINGS DO I SMELL ? ONE
    DO I SEE A LARGE APPLE ? YES.
```

The difficulty of each of these tasks can be modulated by changing the size of the base vocabulary, the length of sequences, or the number of queries. Our particular setting of these parameters will be described in section 6.8.1.

6.4 Standard language classification task

To show the usefulness of measuring WADE on standard language classification tasks, we train various text classifiers on the IMDB dataset and compare their WADE scores with a usual performance metric for classification: accuracy. This classification task, first proposed by (Maass, Natschläger, and Markram 2002), consists of deciding if a movie review is positive or negative from its text. It contains 25000 training examples and 25000 test examples. The two labels are balanced in both the training and test set. This dataset is a high dimensional language-based task with a binary output.

We study five standard models: (i) an Elman recurrent neural network (RNN) (Elman 1990) with tanh activation functions trained with backpropagation through time. (ii) A long-short term memory (LSTM) recurrent neural network (Hochreiter and Schmidhuber 1997), also trained with backpropagation through time. (iii) A gated recurrent unit (GRU) recurrent neural network (Cho et al. 2014). (iv) A standard encoder-only transformer neural network model (Vaswani et al. 2017). (v) A logistic regression using a bag-of-words representation of each sentence as input features. All the

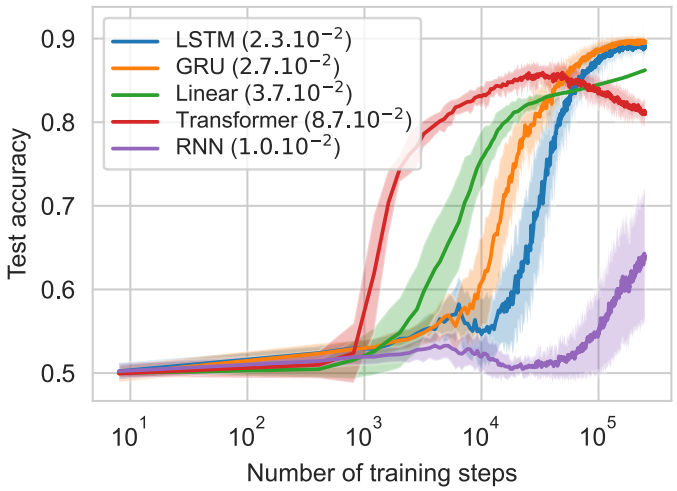


Figure 6.2 – Test accuracy curves for each model. Shaded areas are $\pm 1\sigma$ around the average over 50 runs.

models are trained with batches of training data using the Adam optimization algorithm (Kingma and Ba 2015). Each model's hyperparameters are chosen to ensure they have a similar number of trainable parameters except for the logistic regression whose parameter count is solely determined by the input and output dimensions.

	WADE $\times 10^{-2}$ **(std.)** ↑	**Max test accuracy (std.)** ↑
RNN	1.028 ±0.283	0.705 ±0.076
LSTM	2.280 ±0.303	0.902 ±0.002
GRU	2.711 ±0.318	**0.904** ±0.002
Linear	3.737 ±0.745	0.862 ±0.000
Transformer	**8.716** ±0.720	0.872 ±0.003

Table 6.2 – Comparison of our new WADE metric to assess learning speed and the standard maximum test accuracy on the IMDB classification dataset. Results are averaged over 50 separate runs. (↑ indicates that higher is better).

Table 6.2 shows the results of all models on the IMDB dataset reporting both the standard test accuracy as well as our new WADE metric measuring the learning speed of the different models. We report the maximum test accuracy observed during training. This corresponds to the model checkpoint that would be selected with a validation set before using it on test data. The transformer model learns the fastest of all, as shown by its higher WADE score, but it does not reach test accuracy values as high as the GRU and LSTM model. We think that this could be attributed to the transformer model over-fitting on the available data as discussed below.

According to the WADE metric, the RNN is the worst (i.e., slowest to learn) model

with a score of 1.04×10^{-2} whereas the linear model and the transformer are the fastest with a score above 8×10^{-2}. The transformer seems to have more reliable learning speeds than the linear model as shown by the lower standard deviation (0.7×10^{-2} versus 4.3×10^{-2}). The LSTM and GRU models have intermediate and stable learning speeds. The GRU has slightly better WADE score than the LSTM, which can be also seen by inspecting the learning curves in figure 6.2 where the GRU is consistently above the LSTM.

However, when we look at the maximum test accuracy we obtain a different ordering, with LSTM and GRU performing best with slightly above 90% accuracy followed by the transformer and the linear model (around 86%) and finally the RNN with the lowest accuracy. The LSTM, GRU and linear models all seem to have converged at the end of the experiment. The transformer's lower final accuracy despite it being considered a state of the art model may be explained by over-fitting (apparent on the graph with the test accuracy score going down). Moreover, the RNN clearly hasn't converged at the end of the experiment which also explains the relatively low max accuracy score.

Our new WADE metric gives a complementary view of a model's abilities, which may be significantly different from what can be obtained from the usual performance metrics focused on the final accuracy of the model.

6.5 Reservoir computing and reservoir cellular automata

We use names adapted from (Jaeger 2012) in this section. An echo-state network is a random recurrent neural network with frozen weights and skip-connections. Its random weights are sampled in a specific way so that it performs random combinations of input vectors with its current state while keeping a history of past inputs (Jaeger 2001).

Reservoir cellular automaton (ReCA) is a model similar to echo-state networks but where a cellular automaton (CA) replaces the random RNN. The cellular automaton can be seen as a RNN with additional weigth-sharing similar to convolutional neural network. The special structure of a recurrent CA update makes it more likely than random RNNs to generate complex structures (Wolfram 1983, 2002). Because CAs were not designed to make use of inputs or produce outputs, we extend the model to make it accept input vectors and to make reading from the CA state possible (details in Section 2.3.3).

We also propose two novel projection schemes that offer more flexibility than existing ones. They extend the one-to-one projection and allow for a wider variety of produced encodings:

One-to-many. Each input bit is assigned exactly to k positions in P instead of one. This is equivalent to adding k separate and mutually exclusive one-to-one projections into a single one.

One-to-pattern. Each input bit is assigned to a random contiguous pattern of k bits set in a fixed position.

The effect of applying our three methods to a simple input with three components is illustrated in Fig. 6.3. Projections are chosen to map each input bit to a unique output configuration.

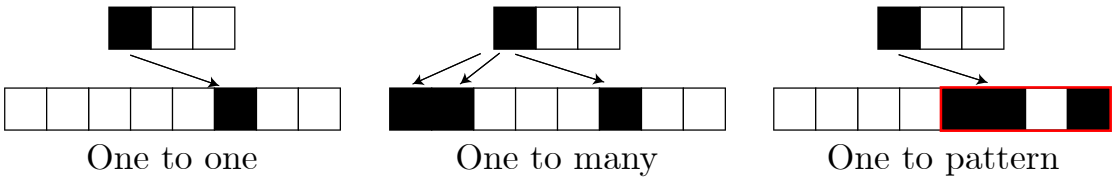


Figure 6.3 – Three encoding methods. From left to right: *one-to-one*, *one-to-many* and *one-to-pattern*.

Following (Nichele and Gundersen 2017; Nichele and Molund 2017a; Yilmaz 2014), the input is projected multiple times to add redundancy to the encoding. This was experimentally observed to improve performance. We also use a projection vector larger than the initial one. Projections are generated randomly R times, applied, and concatenated as a single large encoder to create the full CA input vector. The parameter R is called *redundancy*.

For cellular automata in 1D, the space $\mathcal{P}$ is one dimensional, and the concatenation is performed on this dimension. In higher-dimensional spaces, this concatenation must be defined in another way.

We write the R projection functions $E_1, E_2, \ldots, E_R$. The final size of the input vector and the CA state grid is now $R \times L_d$ instead of L_d. We have

$$\boldsymbol{p}_t = P(\boldsymbol{x}_t) = E_1(\boldsymbol{x}_t) \parallel \ldots \parallel E_R(\boldsymbol{x}_t), \tag{6.4}$$

where $\|$ is the concatenation operator.

For example, other projections have been proposed and implemented in (Yilmaz 2014). We choose to present the three above because they are simple to understand and implement, and their effects can be explored thoroughly.

Input combination. There are multiple ways to combine the input vector with the current state of the CA. Glover et al. (2021) propose to use a XOR function between the projected input and the state of the CA.

At each input step t, the projected input vector $\boldsymbol{p}_t$ is XORed element-wise with the current CA state $\boldsymbol{s}_t$, so that each 1 bit of the input switches the corresponding bit in the CA state to another value. This creates a new state $\boldsymbol{s}'_t$ from which the CA will evolve. $\boldsymbol{s}'_t$ is defined as

$$\boldsymbol{s}'_t \coloneqq \boldsymbol{p}_t \otimes \boldsymbol{s}_t, \tag{6.5}$$

where $\otimes$ is the element-wise XOR operator between two vectors of binary values, and $\boldsymbol{s}'_t$ is the temporary CA state resulting from the combination of the CA state and the input vectors at time t.

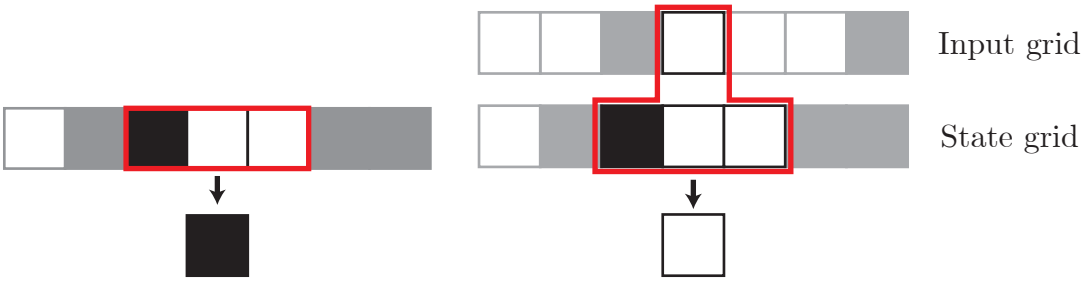
From the state of the CA at time t, $\boldsymbol{s}'_t$, we then compute the next CA states applying the update function Φ,

$$\boldsymbol{s}_{t+1} = \Phi(\boldsymbol{s}'_t). \tag{6.6}$$

This is one of many possible ways to combine an input vector with the current state of the CA that may affect its performance as a reservoir. The XOR-based input combination gives an asymmetrical role to 0 and 1 in the input vectors, which is natural considering how we encode and project categorical data points. Different rules of CAs might benefit or lose performance due to these encoding choices. The combination problem can be generalized to incorporate it into the CA rule search problem. We demonstrate this in the next paragraph.

Generalization. We generalize the input combination method above by incorporating it into the CA update rule. Since the input vector $\boldsymbol{p}$ is binary, we can treat its value at

position i as just another virtual CA cell. For a standard ECA, the update step takes into account the three immediate neighbors $\boldsymbol{s}^{(i-1)}, \boldsymbol{s}^{(i)}, \boldsymbol{s}^{(i+1)}$ plus the additional virtual cell $\boldsymbol{p}^{(i)}$. Instead of 8 possible input configurations for a binary 1D CA, we now have 16. In this extended CA space, the number of possible rules is $2^{16} = 65536$ instead of 256.



(a) Regular CA update only takes neighbor states into account.

(b) The extended neighborhood update also uses an input value to decide the next state of a cell.

Figure 6.4 – Comparison of standard CA update rule and our extended input combination rule.

This method is illustrated in Fig. 6.4 during the regular CA update. It generalizes the concept of input combination with the state $\boldsymbol{s}$. The XOR method described above is one of its special cases, but our method also contains all possible Boolean functions with 4 inputs.

6.6 New benchmark: compared methods

In the following sections, we carry out experiments to measure WADE on the benchmark described in Table 6.1 and introduced in section 6.3.2. We compare several baseline models to understand their learning efficiency: three of the most common sequential machine learning models for which all parameters are trained, RNNs, LSTMs, and transforms, with two methods using reservoir computing: the echo-state networks (ESN, with a random RNN reservoir) and reservoir cellular automata (ReCA, with a cellular automaton reservoir) for which only a fraction of the parameters are learned. Table 6.3 presents all the methods we study in our experiments.

In all our tasks, the input data is assumed to be categorical and sequential. Tokens are observed one by one in sequence, we define this input as $X = [X_1, \ldots, X_t, \ldots]$, $t \in \mathbb{N}$, $\forall t\ X_t \in \mathcal{X} \subset \mathbb{N}$, where t is the time index of the sequence, each X_i is a token corresponding to time index i, and $\mathcal{X}$ is the set of numbered input categories — or

Methods		Type
RNN	Recurrent neural networks	Fully trained
LSTM	Long-short term memory networks	
Transformer		
ESN	Echo-state networks	Reservoir-based
ReCA	Reservoir Cellular Automata	

Table 6.3 – Summary of the compared models: three are fully trained, and two are reservoir-based.

different tokens in the vocabulary. First, each categorical input vector is one-hot encoded into a vector of size L, where $L = |\mathcal{X}|$ is the size of the input vocabulary. We define $\boldsymbol{x}_t$ as the encoded vector form of X_t, and we have $\forall t,\ \boldsymbol{x}_t \in \{0,1\}^L$, with $\sum_{i=1}^{L} (\boldsymbol{x}_t)_i = 1$. Since the tasks are designed to be generally compatible with language modeling, the input and output vocabularies are the same.

6.6.1 Fully trained sequential models (RNN, LSTM and Transformer)

We first study two standard supervised recurrent models: (i) an Elman recurrent neural network (RNN) (Elman 1990) with tanh activation functions trained with back-propagation through time. (ii) A long-short term memory (LSTM) recurrent neural network (Hochreiter and Schmidhuber 1997), also trained with backpropagation through time. (iii) An encoder-only transformer model with positional encoding (Vaswani et al. 2017).

The three models are trained with a batched Adam optimization algorithm (Kingma and Ba 2015) to minimize a cross-entropy loss function between the predicted and target tokens. No other training device, such as dropout, regularization, or normalization, is used for these fully trained baselines.

6.6.2 Echo-state networks and reservoir cellular automata

In our experiments, we also use two reservoir based models. The first is an echo state network (ESN). We compare this model with reservoir cellular automata (ReCA, see Sections 2.3.3 and 6.5 for more details).

6.6.3 Experimental set-up

We ran 100 separate experiments with different random seeds for each of the CA rules, the RNN, LSTM, Transformer, and ESN on each task in the benchmark. These experiments have a separate input projection matrix for the CA, different random weights for the ESN, and different weight initialization for the supervised baselines. The task inputs are also generated from a new seed for every experiment, but we reuse the seeds for the same experiment on different models to ensure they were trained with the same data. Intervals of one standard deviation for these multiple experiments are reported in the result graphs. The code to reproduce our experiments is available on GitHub [†].

Training parameters

For each experiment, we generate 1200 random examples from the task generator. We split this set randomly into a training set with 80% of the data — 960 examples — and a test set with the remaining 240 examples. The reservoir is run on each training example for the reservoir-based models, which creates the input features for training the decoder. The sequential supervised models use batches of single sequences with the Adam algorithm (Kingma and Ba 2015). They are trained for ten epochs in total. With reservoir models, only the last linear layer (the decoder) is trained. We minimize the cross-entropy loss with stochastic gradient descent (SGD), doing only a single pass over the 960 training examples.

Every few training steps, we generate the output predictions on the testing set, decode it, and compute the test accuracy for our WADE metric. Supervision is only applied on tokens that can be predicted — similar to masked language modeling, e.g., only the answer token is used in the symbol counting or question answering tasks. Accuracy is also computed for these symbols only.

6.7 Human performance evaluation

The proposed benchmark is relatively simple to understand, and hence one may ask what human performance would be and what WADE values this would correspond to. In particular, language tasks appear to be readily solvable from one's understanding

†. https://github.com/hugcis/benchmark_learning_efficiency

of the English language. However, similar to how words are encoded as vectors before being processed by a neural network, we need to obfuscate the tokens to make the task equivalent. This could be achieved in a number of ways. For example, if we map all available tokens to random letters of the alphabet, the apparently trivial task

```
I DO NOT SEE PAUL . DO I SEE PAUL ? NO
I HEAR JAMES BUT I DO NOT HEAR PAUL AND JOHN . DO I HEAR JAMES ? YES
```

would become significantly less obvious to humans if written

```
w K k A D r K w A D l W
w Z t g w K k Z D G C r K w Z t l H.
```

One would need to read through several of these sentences to identify patterns such as the fact that `W` and `H` represent Yes or No or that `r` plays the role of a full stop.

We apply the following procedure in our human evaluation experiments:

1. Choose a random task among 10.
2. Apply a random mapping from the task token to letters of the alphabet.
3. Present the sequences one by one with the tokens to be predicted hidden.
4. The user enters his predicted answer.
5. The valid sequence is shown so that the user can learn from their mistakes.
6. Steps 3-5 are repeated until 10 sequences are correctly solved in a row. Then we go back to step 2 with a new task.

We add the requirement that no external device such as pen and paper or a note-taking program should be used during the experiment, so the user relies solely on their memory. The accuracy is computed by counting the number of right answers in a row. For example, three right answers in a row would correspond to an accuracy of 30%, while ten right answers in a row are 100%, which also corresponds to a change of task. This accuracy score is attributed to the first correct answer of the series of correct answers. This means that a series of five correct answers, starting from Question 6 to Question 11, will correspond to reaching the accuracy of 50% in Question 6. This ensures that ten correct answers on the first attempt yield a WADE score of 1.

6.8 Experimental parameters

We report all the parameters used in our experiments in table 6.4:

	Reservoir-based	**Fully trained**
Total number of sequences per task	1200	1200
Number of training sequences	960	960
Number of testing sequences	240	240
Passes over the data (epochs)	1	10
Random runs per task	100	100
Algorithm	SGD	Adam (Kingma and Ba 2015)
Learning rate	0.001	0.001
Regularization	weight decay 0.001	None
Internal state (hidden) size	1800	h task-dependent
Number of trainable parameters	1800 × dictionary_size	$h^2 + 2 \times h \times$ dictionary_size for the RNN[2]
Output non-linearity	Softmax	Softmax
Internal non-linearity	Not applicable	tanh
Batch size	1	1

Table 6.4 – Experimental parameters common to all tasks. Some of the sizes (such as h) vary from task to task.

6.8.1 Task generation parameters

The tasks are generated according to the procedures described in Section 6.3. They have some parameters that allow the difficulty of the task to be adjusted. We report results on ten tasks. The parameters chosen for our experiments are listed below:

Periodic (1) and Increasing period (2): Sequences are generated from patterns of length between 1 and 10.

Easy symbol counting (3): Available symbols are `A`, `B` and `C`. Generated sequences have between 1 and 10 of these symbols in the prompt, and the query has either one, two, or all three symbols.

Hard symbol counting (4): Available symbols are the same as for the previous task. Generated sequences have between 1 and 45 symbols with separators in the

prompt. There is a minimum of 1 query and can be as many queries as there are different patterns in the prompt.

Question answering (5): There are five available names and two verbs. The prompt consists of between one and five names, and the question is about one of these names.

Harder question answering : There are eleven available names and five verbs. The prompt is made of between one and five names, and the question is about one of these names.

Question answering with world (7), with counting (8): There are thirteen available names and seven verbs.

Adjective question answering (9), with counting (10): There are eight available names, six verbs, four color adjectives, and five size adjectives. The prompt consists of between one and six statements, and there are up to eight questions.

We chose relatively small values of these parameters because we are interested in simple tasks so that models can learn from less than 100 examples. Moreover, the number of possible sentences generated from these small values is already huge, with more than 10^{35} possible sentences for task 4, for example. These parameters could also be varied dynamically to change the task's difficulty when needed or turn each task into multiple subtasks or levels of difficulty. We chose the ten separate tasks/settings pairs listed above to get a broad overview of our benchmarked models on a range of task difficulties.

6.9 Results

We report the final weighted average data efficiency (WADE) scores on our benchmark in Table 6.5 and the accuracy results for comparison in Table 6.6. We include the best elementary reservoir cellular automaton (ReCA) and the echo state network (ESN) with the same internal state size for each task.

Figure 6.5 shows the learning curves for the best reservoir cellular automaton, as well as the echo-state network, the recurrent neural network (RNN), LSTM, and the

†. h is chosen to match the number of parameters of the reservoir-based methods.

Task ID - Name	Reservoir		Human subject
	ReCA	**ESN**	
1 - Periodic	**0.78** ±0.05	0.74 ±0.05	1.00
2 - Incremental periodic	0.57 ±0.02	**0.72** ±0.02	1.00
3 - Symbol counting	**0.06** ±0.01	0.04 ±0.01	0.11
4 - Pattern counting	0.12 ±0.04	**0.14** ±0.03	0.09
5 - Basic question answering (QA)	**0.31** ±0.08	0.26 ±0.03	0.15
6 - Harder QA	**0.35** ±0.11	0.26 ±0.03	0.12
7 - QA with world def.	**0.32** ±0.10	0.27 ±0.04	—
8 - QA with world def. & counting	0.09 ±0.03	**0.14** ±0.06	—
9 - Adjective QA	0.04 ±0.03	0.04 ±0.04	—
10 - Adjective QA & counting	0.04 ±0.02	**0.06** ±0.02	—
	Fully supervised		
	RNN	**LSTM**	**Transformer**
1	0.28 ±0.07	0.31 ±0.04	0.31 ±0.06
2	0.32 ±0.11	0.49 ±0.15	0.42 ±0.16
3	0.04 ±0.02	0.03 ±0.01	0.05 ±0.02
4	0.10 ±0.05	0.08 ±0.03	0.13 ±0.04
5	0.17 ±0.08	0.16 ±0.07	0.20 ±0.06
6	0.16 ±0.07	0.12 ±0.06	0.24 ±0.05
7	0.17 ±0.07	0.11 ±0.06	0.24 ±0.05
8	0.10 ±0.05	0.05 ±0.02	0.02 ±0.01
9	**0.05** ±0.03	0.03 ±0.02	0.02 ±0.01
10	0.05 ±0.02	0.03 ±0.02	0.01 ±0.01

Table 6.5 – Comparison of WADE scores (also shown in parentheses in the legend in figure 6.5, higher is better) of the best cellular automaton rules (**ReCA**, Yilmaz 2014) for each task against an echo-state network (**ESN**, Jaeger 2001), a **RNN**, a **LSTM** (Hochreiter and Schmidhuber 1997), and a **Transformer** (Vaswani et al. 2017) with the same number of parameters. The dashed line "—" indicates that the task was too difficult to complete from memory alone for the human subject. The dots indicate that the task name is the same as in the first line. Accuracy scores are also reported in Table 6.6.

Transformer for all ten tasks. Interestingly, the reservoir-based models are consistently more efficient learners than the fully supervised methods, reaching better accuracy in much fewer training steps. For example, for tasks 5, 6, and 7, the LSTM needs ten times more steps to approach the accuracy that the reservoir CA model reached in less than 1000 training steps.

Echo-state networks (ESN) and reservoir cellular automata (ReCA) appear to learn at similar rates, with no clear advantage for one or the other, as each model outperforms the others in five tasks out of ten. On tasks 2, 4, 8, and 10, the ESN is visibly faster than the ReCA (see curves Figure 6.5), which explains the ESN's higher WADE scores even though the ReCA reaches higher accuracy values after several more training steps.

Task ID - Name	Reservoir	
	ReCA	**ESN**
1 - Periodic	0.99 ±0.01	**1.00** ±0.01
2 - Incremental periodic	0.88 ±0.01	0.87 ±0.03
3 - Symbol counting	0.80 ±0.02	0.30 ±0.02
4 - Pattern counting	0.56 ±0.01	0.59 ±0.02
5 - Basic QA	0.81 ±0.03	0.73 ±0.03
6 - Harder QA	0.72 ±0.04	0.57 ±0.03
7 - QA with world def.	0.66 ±0.03	0.60 ±0.03
8 - QA with world def. & counting	0.59 ±0.03	0.56 ±0.03
9 - Adjective QA	0.62 ±0.04	0.49 ±0.03
10 - Adjective QA & counting	0.54 ±0.02	0.46 ±0.03

	Fully supervised		
	RNN	**LSTM**	**Transformer**
1	0.79 ±0.08	0.68 ±0.04	0.69 ±0.04
2	0.87 ±0.00	**0.89** ±0.02	0.88 ±0.00
3	0.33 ±0.03	0.36 ±0.03	**0.97** ±0.01
4	0.61 ±0.05	**0.61** ±0.02	0.54 ±0.03
5	0.51 ±0.02	0.75 ±0.05	**1.00** ±0.00
6	0.51 ±0.03	0.68 ±0.07	**1.00** ±0.00
7	0.51 ±0.04	0.64 ±0.07	**1.00** ±0.00
8	0.47 ±0.05	0.50 ±0.06	**0.98** ±0.01
9	0.43 ±0.04	0.44 ±0.03	**0.87** ±0.11
10	0.47 ±0.04	0.49 ±0.03	**0.57** ±0.04

Table 6.6 – Comparison of accuracy scores (higher is better) of the **ReCA**, **ESN**, **RNN**, **LSTM**, and **Transformer** models with a similar number of parameters. In contrast to the results of table 6.5, the fully-supervised models are more performant when we measure the accuracy, as shown here, but do not necessarily do as well when we measure the speed of learning, as shown in table 6.5.

Even if they lack slightly in accuracy, as seen for example on the curves of task 4,

the two reservoir-based methods consistently outperform the fully supervised sequential methods (RNNs and LSTMs) in terms of learning speeds. This better learning efficiency could be explained by the internal state structure introduced by the CA rules and the special form of the ESN random matrix which favors memory retention whereas usual recurrent network initialization does not — we initialize the weights of the fully trained models from a uniform distribution $\mathcal{U}(-\sqrt{h^{-1}}, \sqrt{h^{-1}})$ in our experiments, where h is the hidden size.

The human subject scores are estimated based on the authors' performance on obfuscated versions of the tasks — all input tokens are mapped to random symbols. These scores may appear surprisingly low in some of these experiments. Although the task examples presented in Section 6.3.2 seem easy to understand. We rely heavily on our prior knowledge about the symbols to understand the patterns — the words in tasks 5–10 or the numbers in tasks 3 and 4. These symbols are randomly remapped in our human experiments, and more examples, as well as a good memory, are needed to understand the tasks and learn the mapping itself, hence the slower learning.

It is interesting to note that RNNs also seem to perform better than LSTMs and Transformers at the beginning of the training in all tasks but the first two. Even though the Transformer is nowadays generally considered a superior model, vanilla RNNs may still remain competitive in the very low data and computation regimes. This is especially pronounced for tasks 8, 9, and 10, where the RNN test accuracy curve starts increasing significantly 1000 steps before the LSTM. After several epochs, the Transformer still outperforms other alternatives in terms of test accuracy on most tasks.

We note that we implicitly assume a fixed cost per training iteration in our experiments and that each training example is seen once. Without these requirements, one could achieve higher WADE results at the cost of additional computations and memory usage by using *replay*-inspired methods (Gepperth and Karaoguz 2016; Hinton and Plaut 1987; Rebuffi et al. 2017; Robins 1993) that retrain the models with past stored inputs. The results in Table 6.5 used each input sequence once.

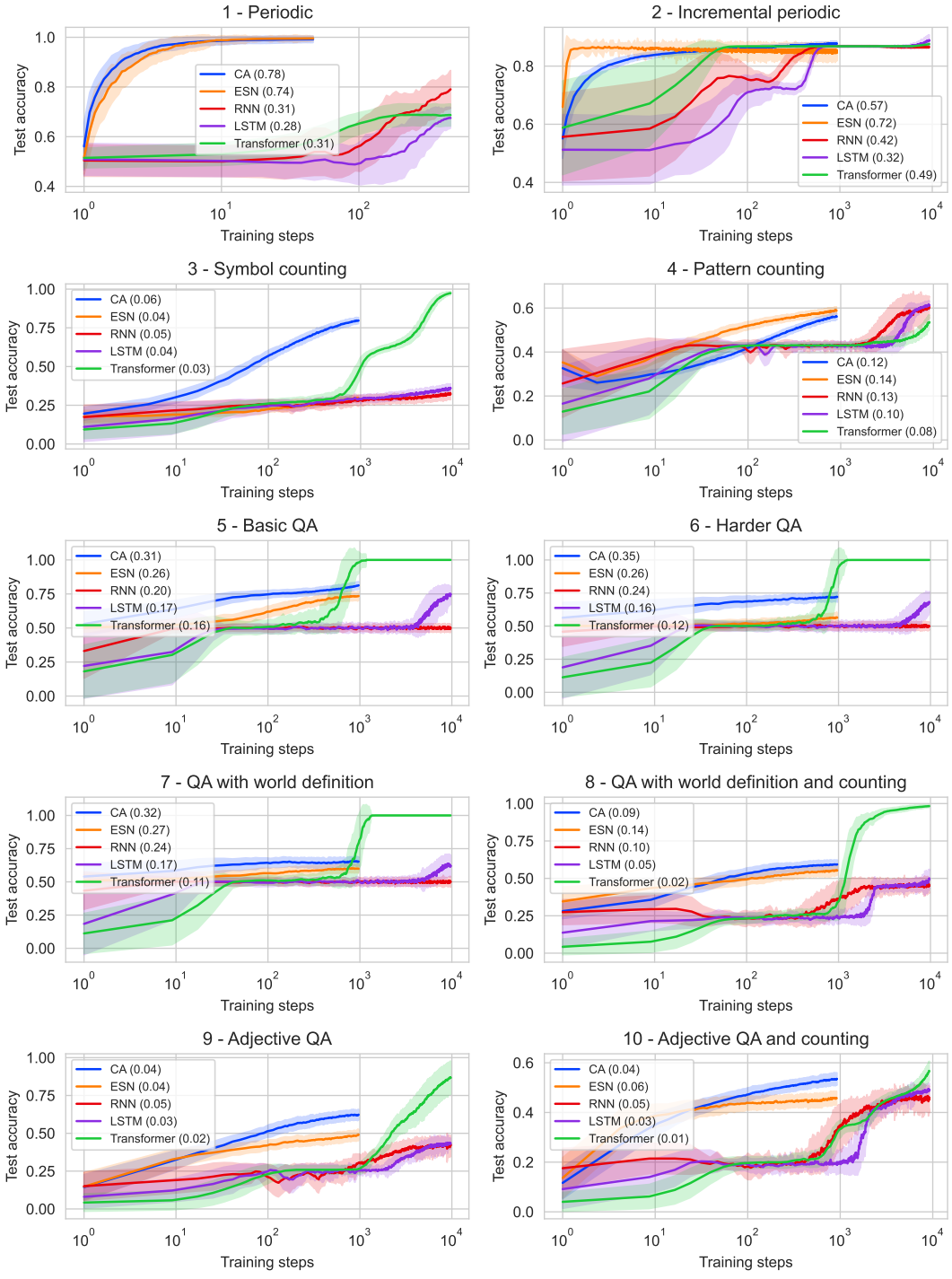


Figure 6.5 – Average learning curves and WADE scores (shown in parentheses in the legend) for each task in the benchmark. The dark blue curves represent the best elementary cellular automata (CA) rule, and the green curves represent the echo-state networks (ESN). Note that the x-axis is logarithmic, showing ten times more training steps for the RNN and LSTM. Shaded areas represent one standard deviation around the average over the 100 different experiments.

6.10 Reservoir cellular automata (ReCA) results

We present some detailed results from our experiments with reservoir cellular automata. First, general statistics about ECA performance on the five first tasks (tasks with ID 1 to 5).

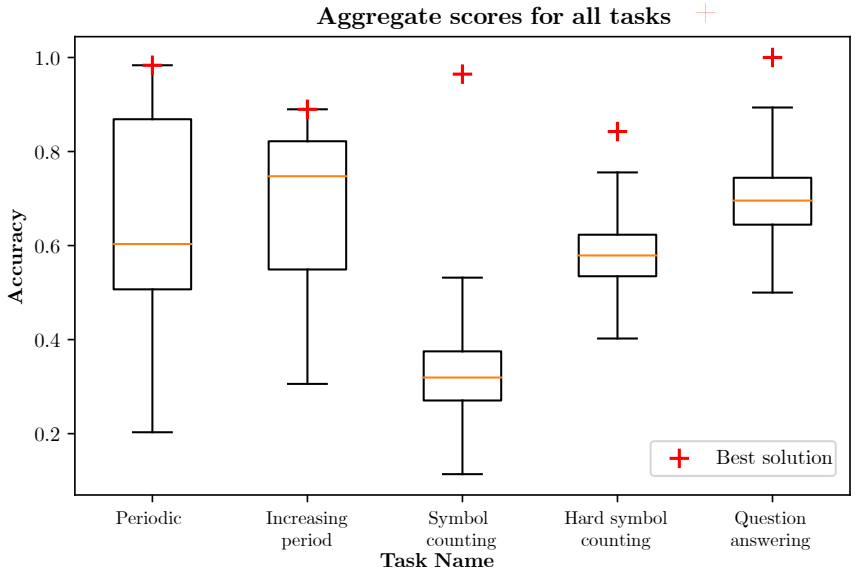


Figure 6.6 – Average test accuracies across rules, hyperparameters, and runs for each task. The best solution for each task is shown in red.

Fig. 6.6 shows a summary of the accuracy scores for each task. Score distributions vary a lot from task to task because of the different output spaces (binary for Periodic, Inc-per, and QA and multi-modal for the others).

Next, we give a more in-depth analysis of the best performing ECAs on the first two tasks.

6.10.1 Binary sequences

The binary sequence task should be the simplest to solve. Since it requires some memory, we expect some CA rules with very chaotic behavior — like ECA rule 30 — to fail at it. We observed that the best rules are consistent with the periodic and ordered types.

The four best rules displayed in figure 6.7 all implement a similar mechanism, using translations of cells from one column to the next at every step. This translation is a memory mechanism, because it shifts any cell that was turned on in reaction to input

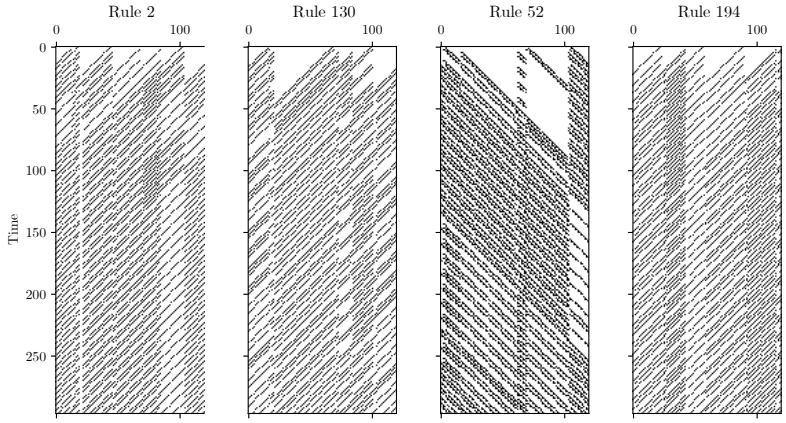


Figure 6.7 – Top performing rules and hyperparameters for the binary periodic sequence task. All of the best rules seem to make use of a lateral translation, which is an effective memory mechanism. Rule 2, 130 and 194 only differ by 1 transition in their rule function.

before it could be overwritten by a subsequent input that would be mapped to the same grid position.

	Top 10 rules				
Rule	38	24	231	152	66
Average accuracy	0.969	0.97	0.97	0.97	0.97
_____	189	194	52	130	2
_____	0.97	0.97	0.97	0.971	0.971

Table 6.7 – Top 10 rules for the periodic binary sequence task (ordered left to right) — best hyper-parameter combination. Many rules get close to the best possible accuracy.

6.10.2 Symbol counting

The easy version of this task yielded the most surprising results: some CA rules unexpectedly reach very good scores. These top rules are reliably found to have the best accuracy scores, even when using powerful decoders. The best performing rule of all the possible ECAs is rule 37.

As shown in Figure 6.8, the two best rules seem to employ a similar "computing mechanism" to solve the task. They seem to grow tree-shaped structures that sometimes become wider. One hypothesis for the success of these two rules is that they encode information about the number of symbols observed in the width of these tree structures. RC is useful for this kind of analysis because it has some level of interpretability. The

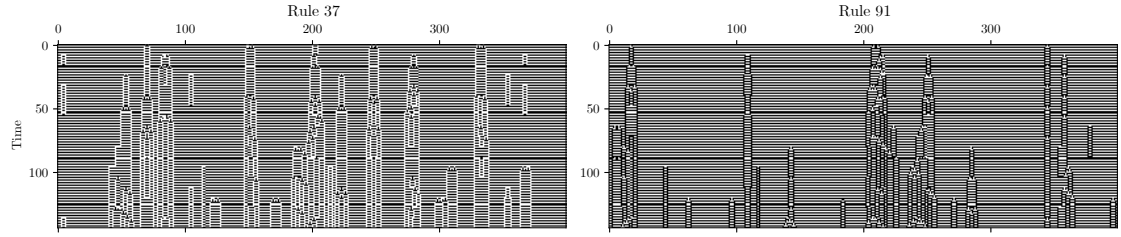


Figure 6.8 – The two best performing rules for the easy symbol counting task (.82 average accuracy).

weights of the output decoding layer can inform us of which of the components of the CAs internal state are the most important for the task being solved. However, even with this tool, it can still be hard to disentangle the effect of each component and their interactions.

	Top 10 rules				
Rule	28	92	220	70	198
Average accuracy	0.571	0.572	0.573	0.574	0.582
_____	156	78	123	91	37
_____	0.594	0.598	0.643	0.721	0.807

Table 6.8 – Top 10 rules for the easy symbol counting task (ordered left to right) — best hyper-parameter combination. Notice the three best rules are largely better than most of the others.

6.11 Conclusions

Learning speed and data efficiency are essential components of any learning system. Our learning speed metric can offer a novel perspective on several machine learning models. Instead of focusing on pure performance, measuring and comparing the data efficiency of different models will hopefully lead to better systems for online and continual learning.

Even with the right metric, it is difficult to thoroughly evaluate the ability of a model to learn efficiently. Our benchmark evaluates a range of problem difficulties which would be challenging to construct by combining or manipulating existing datasets. However, the tasks remain easy to understand and use. Since the tasks are language-based, they can be further mixed or chained to create continuous learning problems and may also be easily extended in a follow-up work.

We study lesser-known machine learning models based on evolving states of complex systems that can learn through self-organization. A complex dynamical system comprises many interacting agents and evolves over time according to a fixed update rule. These agents can be hidden neurons of a RNN or nodes in a graph. Such systems often exhibit emergent global dynamics resulting from the actions of its parts rather than the decisions of a central controller. These dynamics lead to surprisingly complex behavior, which can be random, chaotic, or may lead to unbounded growth of complexity (Nino Boccara 2010). Due to these properties, reservoir computing systems may be a promising alternative that addresses the shortcomings of standard supervised models.

Surprisingly, such models achieve remarkable learning efficiency compared to the more standard sequentially supervised models trained with stochastic gradient-based learning. Complex systems-based models consistently outperform sequential supervised methods and even achieve better learning efficiency than humans on some tasks. They demonstrate more efficient learning on our benchmark at a fraction of the computational and data cost of the conventional models. We believe that more advanced models of this type could lead to more robust and data-efficient machine learning in the future, especially in low-data applications or problems where supervision is limited. Complex systems are underexplored and seem worth further investigating for building the next generation of learning algorithms.

Chapter

7 Conclusion

In this chapter, we summarize our contributions in the thesis.

7.1 Contributions

In this thesis, we have developed some tools to better understand and make use of the computations that occur within complex systems. We summarize our contributions in the following.

— In Chapter 2, we gave an overview of the cellular automaton (CA) model and the specific challenges it poses. We review the deep connection between CAs and recurrent neural networks (RNNs) and how this can be used to apply CAs in reservoir computing (RC). Next, in Chapter 3, we place our work within the vast body of literature on complex systems, complexity, emergence, and learning.

— In Chapter 4, we define a novel metric of complexity for complex systems. The metric is inspired by principles of compression and algorithmic complexity. It uses the ability of small neural networks to learn a model of the local structures in a system, and the evolution of the predictive power of that model as the system evolves over time. We evaluated the quality of this metric by measuring its correlation with human perception of complexity. We find a higher correlation with human annotations than alternative methods on a dataset of CAs rules labeled as complex or not. We use the metric to discover new CAs rules with surprisingly complex behavior semi-automatically.

— In Chapter 5, we built on the work of Chapter 4 to work on the specific challenges posed by working with large-scale systems such as cellular automata. In these systems, qualitatively different behavior may emerge at various scales. To allow dealing with these large systems and apply complexity measures and classification

methods on a range of scales, we developed three coarse-graining methods based on simple statistical analysis, clustering algorithms and autoencoder neural networks that can reduce the size of a system while retaining useful information.

— In Chapter 6, we tackled the issue of using complex systems for general-purpose task solving, using the RC paradigm. We also developed a metric for the speed of learning of machine learning systems. Using that metric, we looked at the data efficiency of some well-known algorithms rather than their task-based performance only. Surprisingly, RC models using random frozen RNNs or CAs are significantly more efficient than other alternatives on a range of tasks. We evaluated this on some standard language datasets and introduced our own dataset of progressively more complex tasks. Efficiency is a crucial property in low data and compute settings, and our work showed that some overlooked methods may actually be competitive in these situations.

Chapter

8 Future Work

This chapter outlines potential future directions for extending our work. Our thesis has investigated the connections between complex systems, open-ended evolution, and learning. While these fields have been studied for some time, many open problems remain, some of them poorly understood, making this research exploratory in nature. We hope that our work can advance our understanding of learning and adaptation in natural and artificial systems, leading to the development of more effective and efficient learning algorithms in the future.

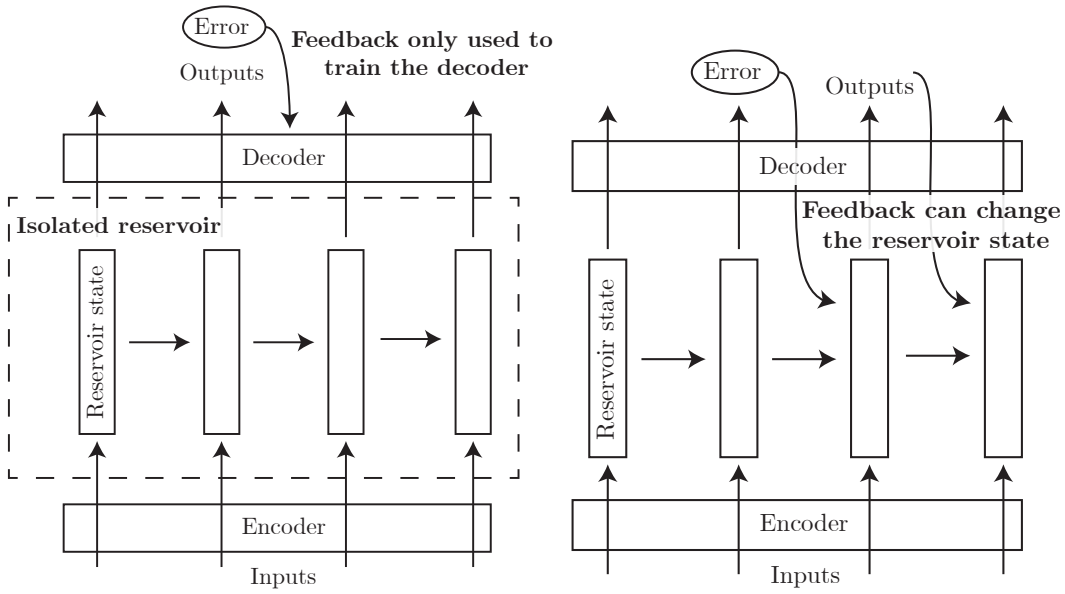
8.1 Evolving cellular automata for complexity

The metric of complexity developed in Chapter 4 and the others presented in Section 3.1 often gives a single value output from a system's evolution over time. Similarly to how Melanie Mitchell, James P Crutchfield, and Das (1996) evolves CAs rules to solve a particular task, it seems appealing to use complexity as a fitness function for a genetic algorithm. Such an algorithm would generate new rules that are progressively more complex (according to the complexity metric).

It is difficult to apply this principle in practice. Because many complexity metrics such as the algorithmic complexity 3.1.6 are approximating some uncomputable ideal value. Candidate systems that maximize a particular metric are often unsatisfying because they exploit a particular flaw of that approximation. This could be accounted for in future work using multiple complementary complexity metrics or using another surrogate metrics, such as the ability to learn to solve auxiliary tasks and generalize from basic tasks to harder ones. This approach motivated our work in Chapter 6, where we design a benchmark of progressively more complex tasks that could be used to evaluate the ability of a system to perform complex computations.

8.2 Feedback reservoir computing

In Chapter 6, we explored the applications of the reservoir computing paradigm to Cellular automata (CAs). In general, RC assumes a purely forward model, which means that inputs are fed into the reservoir and outputs are decoded from it. The trainable decoder layer is trained offline on a training dataset and is then fixed during testing. We illustrate the principle of RC in Figure 8.1a. This structure is often sufficient for supervised tasks when a sufficiently large training dataset is available.



(a) Classical reservoir computing **(b)** Feedback reservoir computing

Figure 8.1 – Comparison of regular reservoir computing with feedback reservoir computing. Time flows from left to right on both Figures.

A straightforward extension that could make the RC systems fully autonomous would be to transmit an error signal within the reservoir state at each output. This extension we call *feedback reservoir computing* is illustrated in Figure 8.1b alongside the standard RC model 8.1a. An ideal system would use an internal learning algorithm embedded within the reservoir state update function and act on the reservoir state, to adjust its future outputs in response to an error signal. This would enable the system to gradually enhance its performance over time. The only mechanism that should be built into this system would be an incentive to minimize the error signals received by adaptation. The main limitation of this idea is that the design of such a system with built-in adaptation is not easy. We do not know of a mechanism that can enable autonomous adaptation

and improvement without external input. The error signal is a type of external input, but it is not specified how the system should use the signal. Our work on complex systems points towards cellular automata and other complex systems as candidates for supporting such a mechanism, which would rely on the growth of complexity.

8.3 Learning in dynamical systems

By examining the learning process in complex systems, neural networks, and the reservoir computing framework from a unified perspective, we can identify a generalization that provides a valuable perspective on the problem of learning in dynamical systems. This generalization allows us to consider the various components of these systems in a more integrated manner, enabling us to develop a more comprehensive understanding of the underlying mechanisms of learning without having to focus on a particular dynamical system.

This form of learning occurs dynamically as a system is simulated or run. Rather than using traditional parameter updates, it takes place in the system's state or "activations", resulting in what we refer to as *learning in dynamical systems*. This process is analogous to an inner loop or "thinking phase", during which the system can self-organize by manipulating its current state without external changes to its parameters. This type of learning is already present in various machine learning systems, including some recurrent neural networks (RNNs), which use short-term dynamical learning to solve language-based tasks (see Section 8.3). Transformers are another type of neural network that has been popular for processing text sequences. The memory of the transformer is not an emergent byproduct of the training process of the neural network because the attention mechanism is directly processing a "window" of input tokens. Emergent memory is also present in some form in the RC framework and in the update rule of gradient-based learning algorithms. Although effective, this form of learning is often not studied or optimized to the same extent as more conventional methods of parameter optimization.

Generic model. We give a general description of a model of learning in dynamical systems. A dynamical system is described by a sequence of variables S_t taking values in some state space $\mathcal{S}$ at times indexed by t. In the most general setting, the state space

can be real-valued or discrete, and the time index may also be continuous or discrete. An evolution function f specifies the state of the dynamical system at time t given a history of states for $t' < t$. It may be deterministic or stochastic.

We study the special case of discrete-time dynamical systems where the current state depends on the previous state.

$$\forall t \in \mathbb{N}, \quad S_{t+1} = f(S_t) \tag{8.1}$$

The inputs $x \in \mathcal{X}$ can be used to condition the update rule and enable the dynamical system to interact with an environment. The space $\mathcal{X}$ can be chosen arbitrarily, for example, the space of real vectors in dimension n, $\mathcal{X} = \mathbb{R}^n$. For an input x, we can write the new dynamical update equation

$$S_{t+1} = f(S_t, x). \tag{8.2}$$

Outputs can be observed from the current state S_t with a decoding function D. We define the output for time step t,

$$y_t = D(S_t). \tag{8.3}$$

Depending on the use cases, alternate representations may be considered where the output depends on two or more time steps or an input vector x. We give a few well-known examples of learning in dynamical systems from the field of machine learning.

Dynamical learning in RNNs We study the example of a question-answering task with a pre-trained RNN. It is easy to see that a RNN can be written fully using equation 8.2 and equation 8.3. The task of answering questions is about reading information from a prompt and answering a question that involves recalling some data from the prompt. An example of such a prompt and question:

$$\underbrace{\texttt{Two cats and five dogs sleep on the red mat.}}_{\text{Prompt}}\underbrace{\texttt{How many dogs are there?}}_{\text{Question}}$$

More advanced question answering requires reasoning and advanced processing of the prompt. In the above example, it would suffice to remember the words of the prompt.

We consider a trained RNN evaluated on a set of question-answering tests. If the data set was designed correctly, the network should not have prior information about a sentence/question pair before reading it. The relevant piece of information of the prompt has to be stored within the values of the hidden state of the RNN and not in the weights of the neural network since they are not changed during this phase. This transient learning that enables the RNN to memorize the beginning of the prompt is exactly the type of dynamical learning that we describe above. RNNs are trained offline to achieve this, but once they are trained, learning happens dynamically.

Modern RNNs are not designed to take full advantage of this memory property and will progressively overwrite the memory if run for more timesteps. Still, they illustrate the type of learning we want to emphasize. We expect advanced systems to be able to keep that memory and also be capable of manipulating stored information to construct a reasoning and form novel concepts.

Supervised learning with stochastic gradient descent. We consider another example: a single hidden layer neural network with weight matrices $W_1 \in \mathbb{R}^{n\times d}$ and $W_2 \in \mathbb{R}^{d\times o}$. We train the network with standard backpropagation and stochastic gradient descent with a learning rate τ. The neural network is represented by a function F, and the loss function between the network output and the label is written as $\mathcal{L}$.

Using the notation of Section 8.3, we define $S_t = \left(W_1^{(t)}, W_2^{(t)}\right) \in \mathbb{R}^{n\times d} \times \mathbb{R}^{d\times o}$ containing the neural network parameters at step t of the training. We assume that the training pairs (X_k, y_k) are numbered according to their selection order for the stochastic gradient descent algorithm. The stochastic gradient update function for the state S_t is:

$$S_{t+1} = f(S_t, X_t, y_t) = \left(W_1^{(t)} - \tau \frac{\partial \mathcal{L}(F(X_t), y_t)}{\partial W_1^{(t)}}, W_2^{(t)} - \tau \frac{\partial \mathcal{L}(F(X_t), y_t)}{\partial W_2^{(t)}}\right). \tag{8.4}$$

Note that the function f is deterministic for a given ordering of training pairs. Therefore, the stochasticity of f is contained in the numbering of training examples.

We note that f is a fixed update function (from the dynamical system point of view of the training pipeline, a training input/output pair (X, y) is a pair of input values). In supervised learning, the function f causes the state to converge to a fixed point. This property is desired when solving a target task characterized by a loss $\mathcal{L}$. Because of

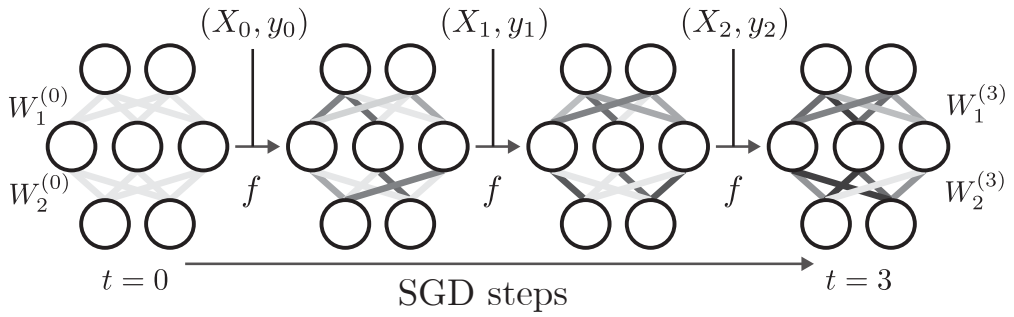


Figure 8.2 – Learning happens within the "activations" of this dynamical system, which in this example are the weights of a neural network being updated with the stochastic gradient descent algorithm (SGD). The fixed update function depends on an input pair (X, y) and the previous state vector $S = (W_1, W_2)$.

that convergence property, a neural network cannot be expected to produce any novel behavior after the end of its training. On the contrary, we know that other complex dynamical systems that we study in this thesis, like cellular automata, may exhibit a behavior of increasing complexity during their development, with no sign of convergence towards a stable state.

Bibliography

Abelson, Harold, Don Allen, Daniel Coore, Chris Hanson, George Homsy, Thomas F. Knight Jr, Radhika Nagpal, Erik Rauch, Gerald Jay Sussman, and Ron Weiss (2000). « Amorphous Computing ». In: *Communications of the ACM* 43.6, pp. 74–82 (Cited on pages 62, 69).

Adams, Alyssa, Hector Zenil, Paul C. W. Davies, and Sara Imari Walker (Apr. 20, 2017). « Formal Definitions of Unbounded Evolution and Innovation Reveal Universal Mechanisms for Open-Ended Evolution in Dynamical Systems ». In: *Scientific Reports* 7.1, p. 997. ISSN: 2045-2322. DOI: 10.1038/s41598-017-00810-8 (Cited on page 31).

Allen, Peter M. and Mark Strathern (2003). « Evolution, Emergence, and Learning in Complex Systems ». In: *Emergence* 5.4, pp. 8–33 (Cited on page 115).

Alomar, M. L., V. Canals, V. Martínez-Moll, and J. L. Rosselló (Sept. 2014). « Low-Cost Hardware Implementation of Reservoir Computers ». In: *2014 24th International Workshop on Power and Timing Modeling, Optimization and Simulation (PATMOS)*. 2014 24th International Workshop on Power and Timing Modeling, Optimization and Simulation (PATMOS), pp. 1–5. DOI: 10.1109/PATMOS.2014.6951899 (Cited on page 37).

Anderson, P. W. (Aug. 4, 1972). « More Is Different ». In: *Science* 177.4047, pp. 393–396. ISSN: 0036-8075, 1095-9203. DOI: 10.1126/science.177.4047.393. pmid: 17796623 (Cited on page 93).

Andre, David, Forrest H. Bennett, and John R. Koza (July 28, 1996). « Discovery by Genetic Programming of a Cellular Automata Rule That Is Better than Any Known Rule for the Majority Classification Problem ». In: *Proceedings of the 1st Annual Conference on Genetic Programming*. Cambridge, MA, USA: MIT Press, pp. 3–11. ISBN: 978-0-262-61127-5 (Cited on page 65).

Antonik, Piotr (2018). *Application of FPGA to Real-Time Machine Learning: Hardware Reservoir Computers and Software Image Processing*. Springer Theses. Cham: Springer International Publishing. ISBN: 978-3-319-91052-9 978-3-319-91053-6. DOI: 10.1007/978-3-319-91053-6 (Cited on page 37).

Antonik, Piotr, Nicolas Marsal, Daniel Brunner, and Damien Rontani (Nov. 2019). « Human Action Recognition with a Large-Scale Brain-Inspired Photonic Computer ». In: *Nature Machine Intelligence* 1.11 (11), pp. 530–537. ISSN: 2522-5839. DOI: 10.1038/s42256-019-0110-8 (Cited on page 38).

Antonik, Piotr, Anteo Smerieri, Francois Duport, Marc Haelterman, and Serge Massar (June 19, 2015). « FPGA Implementation of Reservoir Computing with Online Learning ». In: 24th Belgian-Dutch Conference on Machine Learning (Benelearn). Delft, Netherlands (Cited on page 37).

Antunes, Luis, Lance Fortnow, Dieter van Melkebeek, and N. V. Vinodchandran (Apr. 4, 2006). « Computational Depth: Concept and Applications ». In: *Theoretical Computer Science*. Foundations of Computation Theory (FCT 2003) 354.3, pp. 391–404. ISSN: 0304-3975. DOI: 10.1016/j.tcs.2005.11.033 (Cited on page 49).

Antunes, Luís and Lance Fortnow (June 1, 2009). « Sophistication Revisited ». In: *Theory of Computing Systems* 45.1, pp. 150–161. ISSN: 1433-0490. DOI: 10.1007/s00224-007-9095-5 (Cited on page 52).

Arata, Hideki, Yoshiaki Takai, Nami K. Takai, and Tsuyoshi Yamamoto (1999). « Free-Form Shape Modeling by 3D Cellular Automata ». In: *Proceedings Shape Modeling International'99. International Conference on Shape Modeling and Applications*. IEEE, pp. 242–247 (Cited on page 22).

Babson, Neil and Christof Teuscher (Dec. 15, 2019). « Reservoir Computing with Complex Cellular Automata ». In: *Complex Systems* 28.4, pp. 433–455. ISSN: 08912513. DOI: 10.25088/ComplexSystems.28.4.433 (Cited on page 40).

Bäck, Thomas and Hans-Paul Schwefel (1993). « An Overview of Evolutionary Algorithms for Parameter Optimization ». In: *Evolutionary computation* 1.1, pp. 1–23 (Cited on page 5).

Bagley, R.J. and J.D. Farmer (1991). « Spontaneous Emergence of a Metabolism ». In: *Artificial Life II* X, pp. 93–140 (Cited on page 60).

Baldi, Pierre and Kurt Hornik (Jan. 1989). « Neural Networks and Principal Component Analysis: Learning from Examples without Local Minima ». In: *Neural Networks* 2.1, pp. 53–58. ISSN: 08936080. DOI: 10.1016/0893-6080(89)90014-2 (Cited on page 100).

Banzhaf, Wolfgang, Frank D. Francone, Robert E. Keller, and Peter Nordin (1998). *Genetic Programming: An Introduction: On the Automatic Evolution of Computer Programs and Its Applications*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc. 470 pp. ISBN: 978-1-55860-510-7 (Cited on page 55).

Beaulieu, Shawn, Lapo Frati, Thomas Miconi, Joel Lehman, Kenneth O. Stanley, Jeff Clune, and Nick Cheney (Mar. 3, 2020). « Learning to Continually Learn ». arXiv: 2002.09571 [cs, stat] (Cited on page 32).

Beckers, Ralph, Owen E. Holland, and Jean-Louis Deneubourg (2000). « Fom Local Actions to Global Tasks: Stigmergy and Collective Robotics ». In: *Prerational Intelligence: Adaptive Behavior and Intelligent Systems Without Symbols and Logic, Volume 1, Volume 2 Prerational Intelligence: Interdisciplinary Perspectives on the Behavior of Natural and Artificial Systems, Volume 3*. Ed. by Holk Cruse, Jeffrey Dean, and Helge Ritter. Studies in Cognitive Systems. Dordrecht: Springer Netherlands, pp. 1008–1022. ISBN: 978-94-010-0870-9. DOI: 10.1007/978-94-010-0870-9_63 (Cited on page 53).

Bedau, Mark A., Nicholas Gigliotti, Tobias Janssen, Alec Kosik, Ananthan Nambiar, and Norman Packard (Apr. 1, 2019). « Open-Ended Technological Innovation ». In: *Artificial Life* 25.1, pp. 33–49. ISSN: 1064-5462. DOI: 10.1162/artl_a_00279 (Cited on page 58).

Bedau, Mark A., John S. McCaskill, Norman H. Packard, and Steen Rasmussen (2000). « A Less Abstract Artificial Chemistry ». In: *Artificial Life VII: Proceedings of the Seventh International Conference on Artificial Life*. MIT Press, pp. 49–53. ISBN: 978-0-262-29107-1 (Cited on page 61).

Bennett, Charles H. (1988). « Dissipation, Information, Computational Complexity and the Definition of Organization ». In: *Emerging Syntheses In Science*. CRC Press. ISBN: 978-0-429-49259-4 (Cited on page 49).

— (1995). « Logical Depth and Physical Complexity ». In: *The Universal Turing Machine A Half-Century Survey*. Ed. by Rolf Herken. Red. by Rolf Herken. Vol. 2. Vienna: Springer Vienna, pp. 207–235. ISBN: 978-3-211-82637-9 978-3-7091-6597-3. DOI: 10.1007/978-3-7091-6597-3_8 (Cited on pages 49, 75, 115).

Berlekamp, Elwyn R., John Horton Conway, and Richard K. Guy (2001). *Winning Ways for Your Mathematical Plays*. 2nd ed. Natick, Mass: A.K. Peters. 4 pp. ISBN: 978-1-56881-130-7 978-1-56881-142-0 978-1-56881-143-7 978-1-56881-144-4 (Cited on page 108).

Bertschinger, Nils and Thomas Natschläger (July 2004). « Real-Time Computation at the Edge of Chaos in Recurrent Neural Networks ». In: *Neural Computation* 16.7, pp. 1413–1436. ISSN: 0899-7667. DOI: 10.1162/089976604323057443. pmid: 15165396 (Cited on page 37).

Bilotta, Eleonora and Pietro Pantano (Mar. 2011). « ARTIFICIAL MICRO-WORLDS PART II: CELLULAR AUTOMATA GROWTH DYNAMICS ». In: *International Journal of Bifurcation and Chaos* 21.03, pp. 619–645. ISSN: 0218-1274, 1793-6551. DOI: 10.1142/S0218127411028672 (Cited on page 74).

Boas, Sonja E. M., Yi Jiang, Roeland M. H. Merks, Sotiris A. Prokopiou, and Elisabeth G. Rens (2018). « Cellular Potts Model: Applications to Vasculogenesis and Angiogenesis ». In: *Probabilistic Cellular Automata: Theory, Applications and Future Perspectives*. Ed. by Pierre-Yves Louis and Francesca R. Nardi. Emergence, Complexity and Computation. Cham: Springer International Publishing, pp. 279–310. ISBN: 978-3-319-65558-1. DOI: 10.1007/978-3-319-65558-1_18 (Cited on page 21).

Boccara, N., J. Nasser, and M. Roger (July 1, 1991). « Particlelike Structures and Their Interactions in Spatiotemporal Patterns Generated by One-Dimensional Deterministic Cellular-Automaton Rules ». In: *Physical Review A* 44.2, pp. 866–875. ISSN: 1050-2947, 1094-1622. DOI: 10.1103/PhysRevA.44.866 (Cited on page 103).

Boccara, Nino (Sept. 9, 2010). *Modeling Complex Systems*. Springer Science & Business Media. 500 pp. ISBN: 978-1-4419-6562-2. Google Books: boUorPmcbKMC (Cited on pages 115, 142).

Booker, Lashon B. (2004). *Perspectives on Adaptation in Natural and Artificial Systems (Proceedings Volume in the Santa Fe Institute Studies in the Sciences of comPlexity.)* New York, NY, USA: Oxford University Press, Inc. ISBN: 978-0-19-516293-6 (Cited on page 72).

Booker, Lashon B., David E. Goldberg, and John H. Holland (1989). « Classifier Systems and Genetic Algorithms ». In: *Artificial intelligence* 40.1-3, pp. 235–282 (Cited on pages 55, 64).

Bottou, Léon, Frank E. Curtis, and Jorge Nocedal (2018). « Optimization Methods for Large-Scale Machine Learning ». In: *Siam Review* 60.2, pp. 223–311 (Cited on page 114).

Boyd, S. and L. Chua (Nov. 1985). « Fading Memory and the Problem of Approximating Nonlinear Operators with Volterra Series ». In: *IEEE Transactions on Circuits and Systems* 32.11, pp. 1150–1161. ISSN: 1558-1276. DOI: 10.1109/TCS.1985.1085649 (Cited on page 36).

Buckman, Jacob, Danijar Hafner, George Tucker, Eugene Brevdo, and Honglak Lee (2018). « Sample-Efficient Reinforcement Learning with Stochastic Ensemble Value Expansion ». In: *Advances in neural information processing systems* 31 (Cited on page 116).

Buliga, Marius and Louis H. Kauffman (July 30, 2014). « Chemlambda, Universality and Self-Multiplication ». In: *Artificial Life 14: Proceedings of the Fourteenth International Conference on the Synthesis and Simulation of Living Systems*, pp. 490–497. DOI: 10.7551/978-0-262-32621-6-ch079. arXiv: 1403.8046 [cs, math] (Cited on page 61).

Buonomano, Dean V. and Michael M. Merzenich (Feb. 17, 1995). « Temporal Information Transformed into a Spatial Code by a Neural Network with Realistic Properties ». In: *Science* 267.5200, pp. 1028–1030. ISSN: 0036-8075, 1095-9203. DOI: 10.1126/science.7863330 (Cited on page 35).

Butera, William Joseph (2002). « Programming a Paintable Computer ». PhD thesis. USA: Massachusetts Institute of Technology (Cited on page 69).

Carlini, Nicholas (Apr. 1, 2020). *Digital Logic Gates on Conway's Game of Life - Part 1.* URL: https://nicholas.carlini.com/writing/2020/digital-logic-game-of-life.html (visited on 11/15/2022) (Cited on page 64).

Ceccatto, H. A. and B. A. Huberman (Jan. 1988). « The Complexity of Hierarchical Systems ». In: *Physica Scripta* 37.1, p. 145. ISSN: 1402-4896. DOI: 10.1088/0031-8949/37/1/021 (Cited on page 49).

Chaitin, Gregory J (1977). « Algorithmic Information Theory ». In: *IBM journal of research and development* 21.4, pp. 350–359. ISSN: 0018-8646 (Cited on page 47).

— (Jan. 1969). « On the Length of Programs for Computing Finite Binary Sequences: Statistical Considerations ». In: *J. ACM* 16.1, pp. 145–159. ISSN: 0004-5411. DOI: 10.1145/321495.321506 (Cited on page 47).

— (Dec. 1987). « Toward a Mathematical Definition of "Life" ». In: *Information, Randomness & Incompleteness*. Vol. Volume 8. World Scientific Series in Computer Science Volume 8. WORLD SCIENTIFIC, pp. 86–104. ISBN: 978-9971-5-0479-3. DOI: 10.1142/9789814434058_0011 (Cited on page 46).

— (1990). *Information, Randomness & Incompleteness: Papers on Algorithmic Information Theory*. Vol. 8. World Scientific (Cited on pages 44, 47, 49).

Chan, Bert Wang-Chak (May 4, 2019). « Lenia - Biology of Artificial Life ». arXiv: 1812.05433 [nlin] (Cited on page 22).

Chang, Hao-Hsuan, Hao Song, Yang Yi, Jianzhong Zhang, Haibo He, and Lingjia Liu (Apr. 2019). « Distributive Dynamic Spectrum Access Through Deep Reinforcement Learning: A Reservoir Computing-Based Approach ». In: *IEEE Internet of Things Journal* 6.2, pp. 1938–1948. ISSN: 2327-4662. DOI: 10.1109/JIOT.2018.2872441 (Cited on page 38).

Channon, Alastair (2003). « Improving and Still Passing the ALife Test: Component-normalised Activity Statistics Classify Evolution in Geb as Unbounded ». In: *Proceed-*

ings of Artificial Life VIII, Sydney, RK Standish, MA Bedau, and HA Abbass,(eds.), MIT Press: Cambridge, MA, pp. 173–181 (Cited on pages 60, 72).

Chevalier-Boisvert, Maxime, Dzmitry Bahdanau, Salem Lahlou, Lucas Willems, Chitwan Saharia, Thien Huu Nguyen, and Yoshua Bengio (2018). « Babyai: A Platform to Study the Sample Efficiency of Grounded Language Learning ». arXiv: 1810.08272 (Cited on page 116).

Cho, Kyunghyun, Bart van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio (2014). « On the Properties of Neural Machine Translation: Encoder–Decoder Approaches ». In: *Proceedings of SSST-8, Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation*, pp. 103–111 (Cited on page 124).

Cisneros, Hugo (Aug. 3, 2021). *Open-Ended Creation of Hybrid Creatures with Neural Cellular Automata*. Hugo Cisneros' blog. URL: https://hugocisneros.com/blog/open-ended-creation-of-hybrid-creatures-with-neural-cellular-automata/ (Cited on pages 67, 68).

Cisneros, Hugo, Tomas Mikolov, and Josef Sivic (Nov. 28, 2022). « Benchmarking Learning Efficiency in Deep Reservoir Computing ». In: *Proceedings of The 1st Conference on Lifelong Learning Agents*. Conference on Lifelong Learning Agents (CoLLAs 2022). Vol. 199. PMLR, pp. 532–547 (Cited on pages 11, 12).

Cisneros, Hugo, Josef Sivic, and Tomas Mikolov (Dec. 2019). « Evolving Structures in Complex Systems ». In: *2019 IEEE Symposium Series on Computational Intelligence (SSCI)*. 2019 IEEE Symposium Series on Computational Intelligence (SSCI). Xiamen, China: IEEE, pp. 230–237. ISBN: 978-1-72812-485-8. DOI: 10.1109/SSCI44817.2019.9002840 (Cited on pages 11, 12, 102, 105, 115).

— (July 1, 2020). « Visualizing Computation in Large-Scale Cellular Automata ». In: *Artificial Life Conference Proceedings*. ALife 2020. Vol. 32. MIT Press, pp. 239–247. DOI: 10.1162/isal_a_00277 (Cited on pages 11, 12, 99).

Cook, Matthew (2004). « Universality in Elementary Cellular Automata ». In: *Complex Systems*, p. 40 (Cited on pages 17, 19, 20, 108).

Coore, Daniel (1999). « Botanical Computing: A Developmental Approach to Generating Interconnect Topologies on an Amorphous Computer ». PhD thesis. Massachusetts Institute of Technology (Cited on page 70).

Crutchfield, J. P. and M. Mitchell (Nov. 7, 1995). « The Evolution of Emergent Computation. » In: *Proceedings of the National Academy of Sciences* 92.23, pp. 10742–10746. ISSN: 0027-8424, 1091-6490. DOI: 10.1073/pnas.92.23.10742 (Cited on page 65).

Crutchfield, James P. (Aug. 1, 1994). « The Calculi of Emergence: Computation, Dynamics and Induction ». In: *Physica D: Nonlinear Phenomena* 75.1, pp. 11–54. ISSN: 0167-2789. DOI: 10.1016/0167-2789(94)90273-9 (Cited on pages 50, 51).

— (Jan. 2012). « Between Order and Chaos ». In: *Nature Physics* 8.1 (1), pp. 17–24. ISSN: 1745-2481. DOI: 10.1038/nphys2190 (Cited on page 50).

Crutchfield, James P. and James E. Hanson (Dec. 1993). « Turbulent Pattern Bases for Cellular Automata ». In: *Physica D: Nonlinear Phenomena* 69.3-4, pp. 279–301. ISSN: 01672789. DOI: 10.1016/0167-2789(93)90092-F (Cited on pages 51, 102).

Crutchfield, James P. and Cosma Rohilla Shalizi (1999). « Thermodynamic Depth of Causal States: Objective Complexity via Minimal Representations ». In: *Physical review E* 59.1, p. 275 (Cited on pages 50, 51).

Crutchfield, James P. and Karl Young (July 10, 1989). « Inferring Statistical Complexity ». In: *Physical Review Letters* 63.2, pp. 105–108. ISSN: 0031-9007. DOI: 10.1103/PhysRevLett.63.105 (Cited on pages 50, 51).

Cully, Antoine and Yiannis Demiris (2017). « Quality and Diversity Optimization: A Unifying Modular Framework ». In: *IEEE Transactions on Evolutionary Computation* 22.2, pp. 245–259 (Cited on page 56).

Curry, Haskell B. (1958). *Combinatory Logic*. Amsterdam: North-Holland Pub. Co. (Cited on page 61).

Daley, R. P. (Nov. 1, 1973). « Minimal-Program Complexity of Sequences with Restricted Resources ». In: *Information and Control* 23.4, pp. 301–312. ISSN: 0019-9958. DOI: 10.1016/S0019-9958(73)80001-4 (Cited on page 48).

Daley, Robert (June 1, 1977). « On the Inference of Optimal Descriptions ». In: *Theoretical Computer Science* 4.3, pp. 301–319. ISSN: 0304-3975. DOI: 10.1016/0304-3975(77)90015-9 (Cited on page 48).

Das, Rajarshi, Melanie Mitchell, and James P. Crutchfield (1994). « A Genetic Algorithm Discovers Particle-Based Computation in Cellular Automata ». In: *Parallel Problem Solving from Nature — PPSN III*. Ed. by Yuval Davidor, Hans-Paul Schwefel, and Reinhard Männer. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, pp. 344–353. ISBN: 978-3-540-49001-2. DOI: 10.1007/3-540-58484-6_278 (Cited on page 65).

De Wolf, Tom and Tom Holvoet (2005). « Emergence Versus Self-Organisation: Different Concepts but Promising When Combined ». In: *Engineering Self-Organising Systems*. Ed. by Sven A. Brueckner, Giovanna Di Marzo Serugendo, Anthony Karageorgos, and Radhika Nagpal. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, pp. 1–15. ISBN: 978-3-540-31901-6. DOI: 10.1007/11494676_1 (Cited on page 52).

De Haan, J. (Dec. 1, 2006). « How Emergence Arises ». In: *Ecological Complexity*. Complexity and Ecological Economics 3.4, pp. 293–301. ISSN: 1476-945X. DOI: 10.1016/j.ecocom.2007.02.003 (Cited on page 53).

Dennett, D. C. (1996). *Darwin's Dangerous Idea: Evolution and the Meanings of Life*. 1. Touchstone ed. A Touchstone Book. New York: Simon & Schuster. 586 pp. ISBN: 978-0-684-82471-0 978-0-684-80290-9 (Cited on page 57).

Deutsch, David (1985). « Quantum Theory, the Church–Turing Principle and the Universal Quantum Computer ». In: *Proceedings of the Royal Society of London. A. Mathematical and Physical Sciences* 400.1818, pp. 97–117 (Cited on page 49).

Di Caprio, Dung, Janusz Stafiej, Giorgio Luciano, and Laurent Arurault (2016). « 3D Cellular Automata Simulations of Intra and Intergranular Corrosion ». In: *Corrosion Science* 112, pp. 438–450 (Cited on page 22).

Dittrich, Peter, Jens Ziegler, and Wolfgang Banzhaf (July 2001). « Artificial Chemistries—A Review ». In: *Artificial Life* 7.3, pp. 225–275. ISSN: 1064-5462. DOI: 10.1162/106454601753238636 (Cited on page 61).

Dominey, P., M. Arbib, and J. P. Joseph (1995). « A Model of Corticostriatal Plasticity for Learning Oculomotor Associations and Sequences ». In: *Journal of Cognitive Neuroscience* 7.3, pp. 311–336. ISSN: 0898-929X. DOI: 10.1162/jocn.1995.7.3.311. pmid: 23961864 (Cited on page 37).

Dominey, Peter F. (Aug. 1, 1995). « Complex Sensory-Motor Sequence Learning Based on Recurrent State Representation and Reinforcement Learning ». In: *Biological Cybernetics* 73.3, pp. 265–274. ISSN: 1432-0770. DOI: 10.1007/BF00201428 (Cited on page 38).

Donahue, Colin, Cory Merkel, Qutaiba Saleh, Levs Dolgovs, Yu Kee Ooi, Dhireesha Kudithipudi, and Bryant Wysocki (May 2015). « Design and Analysis of Neuromemristive Echo State Networks with Limited-Precision Synapses ». In: *2015 IEEE Symposium on Computational Intelligence for Security and Defense Applications (CISDA)*. 2015 IEEE Symposium on Computational Intelligence for Security and Defense Applications (CISDA), pp. 1–6. DOI: 10.1109/CISDA.2015.7208623 (Cited on page 37).

Doncieux, Stéphane, Alban Laflaquière, and Alexandre Coninx (July 13, 2019). « Novelty Search: A Theoretical Perspective ». In: *Proceedings of the Genetic and Evolutionary Computation Conference*. GECCO '19: Genetic and Evolutionary Computation Conference. Prague Czech Republic: ACM, pp. 99–106. ISBN: 978-1-4503-6111-8. DOI: 10.1145/3321707.3321752 (Cited on page 56).

Dong, Jonathan, Mushegh Rafayelyan, Florent Krzakala, and Sylvain Gigan (Jan. 2020). « Optical Reservoir Computing Using Multiple Light Scattering for Chaotic Systems Prediction ». In: *IEEE Journal of Selected Topics in Quantum Electronics* 26.1, pp. 1–12. DOI: 10.1109/JSTQE.2019.2936281 (Cited on page 38).

Dyson, Freeman J. (1999). *Origins of Life*. Rev. ed. Cambridge [England] ; New York: Cambridge University Press. 100 pp. ISBN: 978-0-521-62668-2 (Cited on page 57).

Efros, A.A. and T.K. Leung (1999). « Texture Synthesis by Non-Parametric Sampling ». In: *Proceedings of the Seventh IEEE International Conference on Computer Vision*. Proceedings of the Seventh IEEE International Conference on Computer Vision.

Kerkyra, Greece: IEEE, 1033–1038 vol.2. ISBN: 978-0-7695-0164-2. DOI: 10.1109/ICCV.1999.790383 (Cited on page 80).

Eigen, Manfred and Peter Schuster (1979). *The Hypercycle*. Berlin, Heidelberg: Springer. ISBN: 978-3-540-09293-3 978-3-642-67247-7. DOI: 10.1007/978-3-642-67247-7 (Cited on page 57).

Elman, Jeffrey L. (1990). « Finding Structure in Time ». In: *Cognitive Science* 14.2, pp. 179–211. ISSN: 1551-6709. DOI: 10.1207/s15516709cog1402_1 (Cited on pages 57, 114, 124, 130).

Eloranta, Kari and Esa Nummelin (Dec. 1992). « The Kink of Cellular Automaton Rule 18 Performs a Random Walk ». In: *Journal of Statistical Physics* 69.5-6, pp. 1131–1136. ISSN: 0022-4715, 1572-9613. DOI: 10.1007/BF01058766 (Cited on page 102).

Evans, S. C., G. J. Saulnier, and Stephen F. Bush (2003). « A New Universal Two Part Code for Estimation of String Kolmogorov Complexity and Algorithmic Minimum Sufficient Statistic ». In: *DIMACS Workshop on Complexity and Inference* (Cited on page 48).

Feder, M., N. Merhav, and M. Gutman (July 1992). « Universal Prediction of Individual Sequences ». In: *IEEE Transactions on Information Theory* 38.4, pp. 1258–1270. ISSN: 1557-9654. DOI: 10.1109/18.144706 (Cited on page 48).

Feldman, David P and James P Crutchfield (Feb. 9, 1998). « Measures of Statistical Complexity: Why? » In: *Physics Letters A* 238.4, pp. 244–252. ISSN: 0375-9601. DOI: 10.1016/S0375-9601(97)00855-4 (Cited on page 50).

Flamm, Christoph, Alexander Ullrich, Heinz Ekker, Martin Mann, Daniel Högerl, Markus Rohrschneider, Sebastian Sauer, Gerik Scheuermann, Konstantin Klemm, Ivo L Hofacker, and Peter F Stadler (Dec. 2010). « Evolution of Metabolic Networks: A Computational Frame-Work ». In: *Journal of Systems Chemistry* 1.1, p. 4. ISSN: 1759-2208. DOI: 10.1186/1759-2208-1-4 (Cited on page 60).

Floyd, Robert W. and L. S. Steinberg (1976). « An Adaptive Algorithm for Spatial Gray Scale ». In: *Journal of the Society for Information Display* 17, pp. 75–77 (Cited on page 98).

Fogel, Lawrence J., Alvin J. Owens, and Michael J. Walsh (1966). *Artificial Intelligence through Simulated Evolution*. Artificial Intelligence through Simulated Evolution. Oxford, England: John Wiley & Sons, pp. xii, 170. xii, 170 (Cited on page 5).

Fontana, W and L W Buss (Jan. 18, 1994). « What Would Be Conserved If "the Tape Were Played Twice"? » In: *Proceedings of the National Academy of Sciences* 91.2, pp. 757–761. DOI: 10.1073/pnas.91.2.757 (Cited on page 61).

Gács, Peter (1986). « Reliable Computation with Cellular Automata ». In: *Journal of Computer and System Sciences* 32.1, pp. 15–78 (Cited on page 66).

— (Apr. 1, 2001). « Reliable Cellular Automata with Self-Organization ». In: *Journal of Statistical Physics* 103.1, pp. 45–267. ISSN: 1572-9613. DOI: 10.1023/A:1004823720305 (Cited on page 66).

Gandin, Ch-A. and Michel Rappaz (1997). « A 3D Cellular Automaton Algorithm for the Prediction of Dendritic Grain Growth ». In: *Acta Materialia* 45.5, pp. 2187–2195 (Cited on page 22).

Gardner, Martin (Oct. 1970). « Mathematical Games ». In: *Scientific American* 223.4, pp. 120–123. ISSN: 0036-8733. DOI: 10.1038/scientificamerican1070-120 (Cited on pages 20, 92).

Garzon, Max and Fernanda Botelho (1993). « Real Computation with Cellular Automata ». In: *Cellular Automata and Cooperative Systems*. Springer, pp. 191–202 (Cited on page 22).

Gell-Mann, Murray (1988). « Simplicity and Complexity in the Description of Nature ». In: *Engineering and Science* 51.3, pp. 2–9 (Cited on page 48).

Gepperth, Alexander and Cem Karaoguz (Oct. 1, 2016). « A Bio-Inspired Incremental Learning Architecture for Applied Perceptual Problems ». In: *Cognitive Computation* 8.5, pp. 924–934. ISSN: 1866-9964. DOI: 10.1007/s12559-016-9389-5 (Cited on page 137).

Gilpin, William (Sept. 9, 2018). « Cellular Automata as Convolutional Neural Networks ». arXiv: 1809.02942 [cond-mat, physics:nlin, physics:physics] (Cited on pages 27, 34, 66).

Glover, Tom Eivind, Pedro Lind, Anis Yazidi, Evgeny Osipov, and Stefano Nichele (2021). « The Dynamical Landscape of Reservoir Computing with Elementary Cellular Automata ». In: *ALIFE 2021: The 2021 Conference on Artificial Life*. MIT Press (Cited on page 128).

Goldberg, David E. (Jan. 1, 1987). « Simple Genetic Algorithms and the Minimal Deceptive Problem ». In: *Genetic Algorithms and Simulated Annealing*. 1st ed. Los Altos, CA: Pitman, pp. 74–88. ISBN: 978-0-273-08771-7 (Cited on page 57).

Goldstein, Jeffrey (2011). « Emergence in Complex Systems ». In: *The sage handbook of complexity and management*, pp. 65–78 (Cited on page 115).

Gomes, Jorge, Pedro Mariano, and Anders Lyhne Christensen (July 11, 2015). « Devising Effective Novelty Search Algorithms: A Comprehensive Empirical Study ». In: *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*. GECCO '15. New York, NY, USA: Association for Computing Machinery, pp. 943–950. ISBN: 978-1-4503-3472-3. DOI: 10.1145/2739480.2754736 (Cited on page 56).

Gomes, Jorge, Paulo Urbano, and Anders Lyhne Christensen (Sept. 1, 2013). « Evolution of Swarm Robotics Systems with Novelty Search ». In: *Swarm Intelligence* 7.2, pp. 115–144. ISSN: 1935-3820. DOI: 10.1007/s11721-013-0081-z (Cited on page 55).

Goodfellow, Ian J., Mehdi Mirza, Xia Da, Aaron C. Courville, and Yoshua Bengio (2014). « An Empirical Investigation of Catastrophic Forgeting in Gradient-Based Neural Networks ». In: *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*. Ed. by Yoshua Bengio and Yann LeCun (Cited on page 117).

Gosper, R.Wm. (Jan. 1984). « Exploiting Regularities in Large Cellular Spaces ». In: *Physica D: Nonlinear Phenomena* 10.1-2, pp. 75–80. ISSN: 01672789. DOI: 10.1016/0167-2789(84)90251-3 (Cited on pages 93, 95).

Grassberger, Peter (Sept. 1986). « Toward a Quantitative Theory of Self-Generated Complexity ». In: *International Journal of Theoretical Physics* 25.9, pp. 907–938. ISSN: 0020-7748, 1572-9575. DOI: 10.1007/BF00668821 (Cited on pages 43, 46, 93).

Grassberger, Peter (July 3, 1989a). « Problems in Quantifying Self-Generated Complexity ». In: *Helvetica Physica Acta* 62, pp. 489–511. DOI: 10.5169/SEALS-116045 (Cited on page 44).

— (1989b). « Randomness, Information, and Complexity ». In: *Proceedings of the 5th Mexican School on Statistical Physics*. Mexican School on Statistical Physics (EMFE). arXiv: 1208.3459 (Cited on page 75).

Graves, Alex, Greg Wayne, and Ivo Danihelka (2014). « Neural Turing Machines ». arXiv: 1410.5401 (Cited on page 116).

Grunwald, Peter (2007). *The Minimum Description Length Principle*. Adaptive Computation and Machine Learning. Cambridge, Mass: MIT Press. 703 pp. ISBN: 978-0-262-07281-6 (Cited on page 76).

Gutowitz, Howard (Dec. 1, 1995). « Cellular Automata and the Sciences of Complexity (Part I): A Review of Some Outstanding Problems in the Theory of Cellular Automata ». In: *Complexity* 1.5, pp. 16–22. ISSN: 1076-2787. DOI: 10.1002/cplx.6130010505 (Cited on page 44).

Gutowitz, Howard A. (1991). « Transients, Cycles, and Complexity in Cellular Automata ». In: *Physical Review A* 44.12, R7881 (Cited on page 19).

Gutowitz, Howard A. and Christopher G. Langton (1988). « Methods for Designing Cellular Automata with" Interesting" Behavior ». In: *CNLS News Letter* (Cited on page 46).

Hammerschmidt, Katrin, Caroline J. Rose, Benjamin Kerr, and Paul B. Rainey (Nov. 2014). « Life Cycles, Fitness Decoupling and the Evolution of Multicellularity ». In: *Nature* 515.7525, pp. 75–79. ISSN: 0028-0836, 1476-4687. DOI: 10.1038/nature13884 (Cited on page 5).

Hamming, R. W. (Apr. 1950). « Error Detecting and Error Correcting Codes ». In: *Bell System Technical Journal* 29.2, pp. 147–160. ISSN: 00058580. DOI: 10.1002/j.1538-7305.1950.tb00463.x (Cited on page 100).

Hanson, James E. and James P. Crutchfield (Mar. 1992). « The Attractor-Basin Portrait of a Cellular Automaton ». In: *Journal of Statistical Physics* 66.5-6, pp. 1415–1462. ISSN: 0022-4715, 1572-9613. DOI: 10.1007/BF01054429 (Cited on pages 95, 102–104).

— (Apr. 15, 1997). « Computational Mechanics of Cellular Automata: An Example ». In: *Physica D: Nonlinear Phenomena*. Lattice Dynamics 103.1, pp. 169–189. ISSN: 0167-2789. DOI: 10.1016/S0167-2789(96)00259-X (Cited on pages 51, 95, 102, 103, 106).

Harao, Masateru and Shoichi Noguchi (1973). « A Consideration on Cellular Automata with Errors ». In: *Comittee on Automaton, IECE Japan, AL* 73, p. 15 (Cited on page 66).

— (Oct. 1, 1975). « Fault Tolerant Cellular Automata ». In: *Journal of Computer and System Sciences* 11.2, pp. 171–185. ISSN: 0022-0000. DOI: 10.1016/S0022-0000(75)80066-3 (Cited on page 66).

Hasslacher, Brosl (1987). « Discrete Fluids ». In: *Los Alamos Science* (Special Issue), pp. 175–217 (Cited on page 53).

Hauser, Helmut, Auke J. Ijspeert, Rudolf M. Füchslin, Rolf Pfeifer, and Wolfgang Maass (Dec. 1, 2011). « Towards a Theoretical Foundation for Morphological Computation with Compliant Bodies ». In: *Biological Cybernetics* 105.5, pp. 355–370. ISSN: 1432-0770. DOI: 10.1007/s00422-012-0471-0 (Cited on page 37).

Herel, David, Dominika Zogatova, Matej Kripner, and Tomas Mikolov (July 18, 2022). « Emergence of Novelty in Evolutionary Algorithms ». In: ALIFE 2022: The 2022 Conference on Artificial Life. MIT Press. DOI: 10.1162/isal_a_00501 (Cited on page 57).

Hinton, Geoffrey E. (Sept. 1989). « Connectionist Learning Procedures ». In: *Artificial Intelligence* 40.1-3, pp. 185–234. ISSN: 00043702. DOI: 10.1016/0004-3702(89)90049-0 (Cited on page 100).

Hinton, Geoffrey E. and David C. Plaut (1987). « Using Fast Weights to Deblur Old Memories ». In: *Proceedings of the Ninth Annual Conference of the Cognitive Science Society*, pp. 177–186 (Cited on page 137).

Hinton, Geoffrey E. and Terrence J. Sejnowski (May 24, 1999). *Unsupervised Learning: Foundations of Neural Computation*. MIT Press. 420 pp. ISBN: 978-0-262-58168-4. Google Books: yj04Y0lje4cC (Cited on page 3).

Hochreiter, Sepp and Jürgen Schmidhuber (Nov. 1, 1997). « Long Short-Term Memory ». In: *Neural Computation* 9.8, pp. 1735–1780. ISSN: 0899-7667. DOI: 10.1162/neco.1997.9.8.1735 (Cited on pages 114, 116, 117, 124, 130, 135).

Holland, John H. (1992). *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*. 1st MIT Press ed. Complex Adaptive Systems. Cambridge, Mass: MIT Press. 211 pp. ISBN: 978-0-262-08213-6 978-0-262-58111-0 (Cited on page 57).

— (2000). *Emergence: From Chaos to Order*. Oxford University Press. 276 pp. ISBN: 978-0-19-286211-2. Google Books: VjKtpujRGuAC (Cited on page 53).

Hopcroft, John E., Rajeev Motwani, and Jeffrey D. Ullman (2007). *Introduction to Automata Theory, Languages, and Computation*. 3rd ed. Boston: Pearson/Addison Wesley. 535 pp. ISBN: 978-0-321-45536-9 978-0-321-46225-1 978-0-321-45537-6 (Cited on page 47).

Hudcová, Barbora and Tomas Mikolov (July 1, 2020). « Classification of Complex Systems Based on Transients ». In: ALIFE 2020: The 2020 Conference on Artificial Life. MIT Press, pp. 367–375. DOI: 10.1162/isal_a_00260 (Cited on pages 18, 19).

— (July 19, 2021). « Computational Hierarchy of Elementary Cellular Automata ». In: ALIFE 2021: The 2021 Conference on Artificial Life. MIT Press. DOI: 10.1162/isal_a_00447 (Cited on page 19).

— (Mar. 16, 2022). « Classification of Discrete Dynamical Systems Based on Transients ». In: *Artificial Life* 27.3–4, pp. 220–245. ISSN: 1064-5462. DOI: 10.1162/artl_a_00342 (Cited on page 18).

Hutton, Tim J. (Jan. 2007). « Evolvable Self-Reproducing Cells in a Two-Dimensional Artificial Chemistry ». In: *Artificial Life* 13.1, pp. 11–30. ISSN: 1064-5462, 1530-9185. DOI: 10.1162/artl.2007.13.1.11 (Cited on page 60).

Igblan - Life Universal Computer (2022). URL: http://www.igblan.free-online.co.uk/igblan/ca/ (visited on 11/02/2022) (Cited on page 20).

Ikegami, Takashi, Yasuhiro Hashimoto, and Mizuki Oka (2019). « Open-Ended Evolution and a Mechanism of Novelties in Web Services ». In: *Artificial Life* 25.2, pp. 168–177. ISSN: 1530-9185. DOI: 10.1162/artl_a_00287. pmid: 31150293 (Cited on page 58).

Ilachinski, Andrew (July 3, 2001). *Cellular Automata: A Discrete Universe*. World Scientific Publishing Company. 842 pp. ISBN: 978-981-310-256-9. Google Books: BPY7DQAAQBAJ (Cited on page 27).

Ingólfsson, Helgi I., Cesar A. Lopez, Jaakko J. Uusitalo, Djurre H. de Jong, Srinivasa M. Gopal, Xavier Periole, and Siewert J. Marrink (May 2014). « The Power of Coarse Graining in Biomolecular Simulations ». In: *Wiley Interdisciplinary Reviews: Computational Molecular Science* 4.3, pp. 225–248. ISSN: 17590876. DOI: 10.1002/wcms.1169 (Cited on page 94).

Ishiguro, Akio, Akinobu Fujii, and Peter Eggenberger Hotz (Mar. 1, 2003). « Neuromodulated Control of Bipedal Locomotion Using a Polymorphic CPG Circuit ». In: *Adaptive Behavior* 11.1, pp. 7–17. ISSN: 1059-7123. DOI: 10.1177/10597123030111001 (Cited on page 32).

Israeli, Navot and Nigel Goldenfeld (Feb. 20, 2004). « On Computational Irreducibility and the Predictability of Complex Physical Systems ». In: *Physical Review Letters* 92.7, p. 074105. ISSN: 0031-9007, 1079-7114. DOI: 10.1103/PhysRevLett.92.074105. arXiv: nlin/0309047 (Cited on page 94).

— (Feb. 6, 2006). « Coarse-Graining of Cellular Automata, Emergence, and the Predictability of Complex Systems ». In: *Physical Review E* 73.2, p. 026203. ISSN: 1539-3755, 1550-2376. DOI: 10.1103/PhysRevE.73.026203. arXiv: nlin/0508033 (Cited on page 94).

Jaeger, Herbert (2001). « The "Echo State" Approach to Analysing and Training Recurrent Neural Networks-with an Erratum Note ». In: *Bonn, Germany: German National Research Center for Information Technology GMD Technical Report* 148.34, p. 13 (Cited on pages 36, 38, 115, 126, 135).

Jaeger, Herbert (Jan. 1, 2002). « Adaptive Nonlinear System Identification with Echo State Networks ». In: *Proceedings of the 15th International Conference on Neural Information Processing Systems*. NIPS'02. Cambridge, MA, USA: MIT Press, pp. 609–616 (Cited on page 38).

— (Feb. 1, 2012). *Long Short-Term Memory in Echo State Networks: Details of a Simulation Study*. Jacobs University Bremen (Cited on pages 39, 40, 126).

Jalalvand, Azarakhsh, Glenn Van Wallendael, and Rik Van De Walle (June 2015). « Real-Time Reservoir Computing Network-Based Systems for Detection Tasks on Visual Contents ». In: *2015 7th International Conference on Computational Intelligence, Communication Systems and Networks*. 2015 7th International Conference on Computational Intelligence, Communication Systems and Networks, pp. 146–151. DOI: 10.1109/CICSyN.2015.35 (Cited on page 38).

Jennings, N. R. (1999). « Agent-Based Computing: Promise and Perils ». In: 16th Int. Joint Conf. on Artificial Intelligence (IJCAI-99) (01/01/99). In collab. with N. R. Jennings, pp. 1429–1436 (Cited on page 62).

Johnson, Neil F. (2009). *Simply Complexity: A Clear Guide to Complexity Theory*. Oxford: Oneworld. 236 pp. ISBN: 978-1-85168-630-8 (Cited on page 59).

Jordan, Michael I. (1997). « Serial Order: A Parallel Distributed Processing Approach ». In: *Advances in Psychology*. Vol. 121. Elsevier, pp. 471–495. ISBN: 978-0-444-81931-4. DOI: 10.1016/S0166-4115(97)80111-2 (Cited on page 57).

Joulin, Armand and Tomas Mikolov (Dec. 7, 2015). « Inferring Algorithmic Patterns with Stack-Augmented Recurrent Nets ». In: *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1*. NIPS'15. Cambridge, MA, USA: MIT Press, pp. 190–198 (Cited on page 117).

Kanazawa, Satoshi (2004). « General Intelligence as a Domain-Specific Adaptation. » In: *Psychological review* 111.2, p. 512 (Cited on page 114).

Kanno, Kazutaka and Atsushi Uchida (Mar. 8, 2022). « Photonic Reinforcement Learning Based on Optoelectronic Reservoir Computing ». In: *Scientific Reports* 12.1 (1), p. 3720. ISSN: 2045-2322. DOI: 10.1038/s41598-022-07404-z (Cited on page 38).

Kari, Jarkko J. (2012). « Basic Concepts of Cellular Automata ». In: *Handbook of Natural Computing*. Ed. by Grzegorz Rozenberg, Thomas Bäck, and Joost N. Kok. Berlin, Heidelberg: Springer, pp. 3–24. ISBN: 978-3-540-92910-9. DOI: 10.1007/978-3-540-92910-9_1 (Cited on page 16).

Kauffman, Stuart (1995). *At Home in the Universe: The Search for Laws of Self-organization and Complexity*. Oxford University Press. 348 pp. ISBN: 978-0-19-509599-9 (Cited on page 53).

Kelly, David, Mark Dillingham, Andrew Hudson, and Karoline Wiesner (2012). « A New Method for Inferring Hidden Markov Models from Noisy Time Sequences ». In: *PloS One* 7.1, e29703. ISSN: 1932-6203. DOI: 10.1371/journal.pone.0029703. pmid: 22247783 (Cited on page 51).

Kingma, Diederik P. and Jimmy Ba (2015). « Adam: A Method for Stochastic Optimization ». In: *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*. Ed. by Yoshua Bengio and Yann LeCun (Cited on pages 125, 130, 131, 133).

Kirby, Kevin G. (1991). « Context Dynamics in Neural Sequential Learning ». In: *Proc. Florida AI Research Symposium (FLAIRS) 1991*. Florida AI Research Symposium, pp. 66–70 (Cited on page 35).

Kistemaker, Steijn and Shimon Whiteson (July 12, 2011). « Critical Factors in the Performance of Novelty Search ». In: *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation*. GECCO '11. New York, NY, USA: Association for Computing Machinery, pp. 965–972. ISBN: 978-1-4503-0557-0. DOI: 10.1145/2001576.2001708 (Cited on page 56).

Kitayama, Ken-ichi, Masaya Notomi, Makoto Naruse, Koji Inoue, Satoshi Kawakami, and Atsushi Uchida (Sept. 9, 2019). « Novel Frontier of Photonics for Data Processing—Photonic Accelerator ». In: *APL Photonics* 4.9, p. 090901. DOI: 10.1063/1.5108912 (Cited on page 38).

Kleyko, Denis, E. Paxon Frady, and Friedrich T. Sommer (Oct. 7, 2020). « Cellular Automata Can Reduce Memory Requirements of Collective-State Computing ». arXiv: 2010.03585 [cs] (Cited on page 40).

Kmiecik, Sebastian, Dominik Gront, Michal Kolinski, Lukasz Wieteska, Aleksandra Elzbieta Dawid, and Andrzej Kolinski (July 27, 2016). « Coarse-Grained Protein Models and Their Applications ». In: *Chemical Reviews* 116.14, pp. 7898–7936. ISSN: 0009-2665, 1520-6890. DOI: 10.1021/acs.chemrev.6b00163 (Cited on page 94).

Ko, Ker-I (Jan. 1, 1986). « On the Notion of Infinite Pseudorandom Sequences ». In: *Theoretical Computer Science* 48, pp. 9–33. ISSN: 0304-3975. DOI: 10.1016/0304-3975(86)90081-2 (Cited on page 48).

Kolmogorov, A. N. (Jan. 1968). « Three Approaches to the Quantitative Definition of Information ». In: *International Journal of Computer Mathematics* 2.1-4, pp. 157–168. ISSN: 0020-7160, 1029-0265. DOI: 10.1080/00207166808803030 (Cited on pages 47, 77).

Koppel, Moshe (1988). « Structure ». In: *The Universal Turing Machine: A Half-Century Survey*. R. Herken. Vol. 2. Oxford University Press, pp. 403–419. ISBN: 978-3-211-82637-9 (Cited on page 51).

Koppel, Moshe and Henri Atlan (Aug. 1, 1991). « An Almost Machine-Independent Theory of Program-Length Complexity, Sophistication, and Induction ». In: *Information Sciences* 56.1, pp. 23–33. ISSN: 0020-0255. DOI: 10.1016/0020-0255(91)90021-L (Cited on pages 51, 115).

Kowaliw, Taras (2008). « Measures of Complexity for Artificial Embryogeny ». In: *Proceedings of the 10th Annual Conference on Genetic and Evolutionary Computation - GECCO '08*. The 10th Annual Conference. Atlanta, GA, USA: ACM Press, p. 843. ISBN: 978-1-60558-130-9. DOI: 10.1145/1389095.1389259 (Cited on page 76).

Koza, J (1992). « Evolution of Subsumption Using Genetic Programming ». In: *Proc. First European Conference on Artificial Life Towards a Practice of Autonomous Systems, 1992*. European Conference on Artificial Life Towards a Practice of Autonomous Systems. MIT Press (Cited on page 65).

Koza, John R. (June 1, 1994). « Genetic Programming as a Means for Programming Computers by Natural Selection ». In: *Statistics and Computing* 4.2, pp. 87–112. ISSN: 1573-1375. DOI: 10.1007/BF00175355 (Cited on page 55).

Kramer, Mark A. (Feb. 1991). « Nonlinear Principal Component Analysis Using Autoassociative Neural Networks ». In: *AIChE Journal* 37.2, pp. 233–243. ISSN: 0001-1541, 1547-5905. DOI: 10.1002/aic.690370209 (Cited on page 100).

Krizhevsky, Alex and Geoffrey E. Hinton (2009). *Learning Multiple Layers of Features from Tiny Images*. University of Toronto (Cited on page 117).

Kruszewski, Germán and Tomas Mikolov (Mar. 16, 2022). « Emergence of Self-Reproducing Metabolisms as Recursive Algorithms in an Artificial Chemistry ». In: *Artificial Life* 27.3–4, pp. 277–299. ISSN: 1064-5462. DOI: 10.1162/artl_a_00355 (Cited on page 61).

Langdon, William B. and Riccardo Poli (2002). *Foundations of Genetic Programming*. Berlin, Heidelberg: Springer. ISBN: 978-3-642-07632-9 978-3-662-04726-2. DOI: 10.1007/978-3-662-04726-2 (Cited on page 55).

Langton, Christopher G. (Jan. 1, 1984). « Self-Reproduction in Cellular Automata ». In: *Physica D: Nonlinear Phenomena* 10.1, pp. 135–144. ISSN: 0167-2789. DOI: 10.1016/0167-2789(84)90256-2 (Cited on pages 60, 73, 92).

— (Oct. 1, 1986). « Studying Artificial Life with Cellular Automata ». In: *Physica D: Nonlinear Phenomena*. Proceedings of the Fifth Annual International Conference 22.1, pp. 120–149. ISSN: 0167-2789. DOI: 10.1016/0167-2789(86)90237-X (Cited on pages 23, 53).

— ed. (1989). *Artificial Life: The Proceedings of an Interdisciplinary Workshop on the Synthesis and Simulation of Living Systems, Held September, 1987, in Los Alamos, New Mexico*. Santa Fe Institute Studies in the Sciences of Complexity v. 6. Redwood City, Calif: Addison-Wesley Pub. Co., Advanced Book Program. 655 pp. ISBN: 978-0-201-09346-9 978-0-201-09356-8 (Cited on page 57).

— (June 1990). « Computation at the Edge of Chaos: Phase Transitions and Emergent Computation ». In: *Physica D: Nonlinear Phenomena* 42.1-3, pp. 12–37. ISSN: 01672789. DOI: 10.1016/0167-2789(90)90064-V (Cited on pages 8, 23, 45, 73, 87).

« Editor's Introduction » (Oct. 1993). In: *Artificial Life* 1 (1_2). Ed. by Christopher G. Langton, pp. v–viii. ISSN: 1064-5462, 1530-9185. DOI: 10.1162/artl.1993.1.1_2.v (Cited on page 71).

Lawrence, Peter A. (1992). *The Making of a Fly: The Genetics of Animal Design*. Oxford [England] ; Cambridge, Mass., USA: Blackwell Science. 228 pp. ISBN: 978-0-632-03048-4 (Cited on page 70).

Legenstein, Robert and Wolfgang Maass (Apr. 1, 2007). « Edge of Chaos and Prediction of Computational Performance for Neural Circuit Models ». In: *Neural Networks*. Echo State Networks and Liquid State Machines 20.3, pp. 323–334. ISSN: 0893-6080. DOI: 10.1016/j.neunet.2007.04.017 (Cited on page 37).

Lehman, Joel and Kenneth O. Stanley (July 7, 2010). « Efficiently Evolving Programs through the Search for Novelty ». In: *Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation*. GECCO '10. New York, NY, USA: Association for Computing Machinery, pp. 837–844. ISBN: 978-1-4503-0072-8. DOI: 10.1145/1830483.1830638 (Cited on page 56).

— (June 2011a). « Abandoning Objectives: Evolution Through the Search for Novelty Alone ». In: *Evolutionary Computation* 19.2, pp. 189–223. ISSN: 1063-6560, 1530-9304. DOI: 10.1162/EVCO_a_00025 (Cited on pages 53, 55, 56).

— (2011b). « Evolving a Diversity of Virtual Creatures through Novelty Search and Local Competition ». In: *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation*, pp. 211–218 (Cited on page 53).

Lempel, A. and J. Ziv (Jan. 1976). « On the Complexity of Finite Sequences ». In: *IEEE Transactions on Information Theory* 22.1, pp. 75–81. ISSN: 0018-9448. DOI: 10.1109/TIT.1976.1055501 (Cited on pages 48, 74).

Levitt, Michael and Arieh Warshel (Feb. 1975). « Computer Simulation of Protein Folding ». In: *Nature* 253.5494, pp. 694–698. ISSN: 0028-0836, 1476-4687. DOI: 10.1038/253694a0 (Cited on page 94).

Li, Wentian and Norman H. Packard (1990). « The Structure of the Elementary Cellular Automata Rule Space ». In: *Complex Systems* 4.3, p. 17 (Cited on page 17).

Li, Wentian, Norman H. Packard, and Christopher G. Langton (Sept. 2, 1990). « Transition Phenomena in Cellular Automata Rule Space ». In: *Physica D: Nonlinear Phenomena* 45.1, pp. 77–94. ISSN: 0167-2789. DOI: 10.1016/0167-2789(90)90175-O (Cited on pages 46, 73).

Liapis, Antonios, Georgios N. Yannakakis, and Julian Togelius (Mar. 1, 2015). « Constrained Novelty Search: A Study on Game Content Generation ». In: *Evolutionary Computation* 23.1, pp. 101–129. ISSN: 1063-6560. DOI: 10.1162/EVCO_a_00123 (Cited on page 56).

Lindgren, Kristian and Mats G. Nordahl (1988). « Complexity Measures and Cellular Automata ». In: p. 32 (Cited on page 46).

Lloyd, Seth and Heinz Pagels (Nov. 1, 1988). « Complexity as Thermodynamic Depth ». In: *Annals of Physics* 188, pp. 186–213. ISSN: 0003-4916. DOI: 10.1016/0003-4916(88)90094-2 (Cited on pages 49, 50).

Lorenz, Dirk M., Alice Jeng, and Michael W. Deem (June 2011). « The Emergence of Modularity in Biological Systems ». In: *Physics of life reviews* 8.2, pp. 129–160. ISSN: 1571-0645. DOI: 10.1016/j.plrev.2011.02.003. pmid: 21353651 (Cited on page 5).

Lukoševičius, Mantas and Herbert Jaeger (2009). « Reservoir Computing Approaches to Recurrent Neural Network Training ». In: *Computer Science Review* 3.3, pp. 127–149 (Cited on page 36).

LuValle, Brian J. (Mar. 28, 2019). « The Effects of Boundary Conditions on Cellular Automata ». In: *Complex Systems* 28.1, pp. 97–124. ISSN: 08912513. DOI: 10.25088/ComplexSystems.28.1.97 (Cited on page 75).

Maas, Andrew L., Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts (2011). « Learning Word Vectors for Sentiment Analysis ». In: *The 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies, Proceedings of the Conference, 19-24 June, 2011, Portland, Oregon, USA*. Ed. by Dekang Lin, Yuji Matsumoto, and Rada Mihalcea. The Association for Computer Linguistics, pp. 142–150 (Cited on pages 115, 116).

Maass, Wolfgang, Thomas Natschläger, and Henry Markram (Nov. 1, 2002). « Real-Time Computing Without Stable States: A New Framework for Neural Computation Based on Perturbations ». In: *Neural Computation* 14.11, pp. 2531–2560. ISSN: 0899-7667, 1530-888X. DOI: 10.1162/089976602760407955 (Cited on pages 36, 124).

— (2004). « Fading Memory and Kernel Properties of Generic Cortical Microcircuit Models ». In: *Journal of Physiology, Paris* 98.4-6, pp. 315–330. ISSN: 0928-4257. DOI: 10.1016/j.jphysparis.2005.09.020. pmid: 16310350 (Cited on page 37).

Mahoney, Matthew V (2000). « Fast Text Compression with Neural Networks ». In: *FLAIRS*. FLAIRS Conference, p. 5 (Cited on pages 74, 80).

Maley, C. C. (July 13, 1999). « Four Steps toward Open-Ended Evolution ». In: *Proceedings of the 1st Annual Conference on Genetic and Evolutionary Computation - Volume 2*. GECCO'99. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., pp. 1336–1343. ISBN: 978-1-55860-611-1 (Cited on page 58).

Margem, Mrwan and Osman Gedik (Dec. 17, 2018). « Reservoir Computing Based on Cellular Automata (ReCA) in Sequence Learning ». In: *Journal of cellular automata* 14, pp. 153–170 (Cited on page 40).

Margem, Mrwan and Ozgür Yilmaz (2017). *An Experimental Study on Cellular Automata Reservoir in Pathological Sequence Learning Tasks*. Jul (Cited on page 40).

Martinenghi, Romain, Sergei Rybalko, Maxime Jacquot, Yanne K. Chembo, and Laurent Larger (June 15, 2012). « Photonic Nonlinear Transient Computing with Multiple-Delay Wavelength Dynamics ». In: *Physical Review Letters* 108.24, p. 244101. DOI: 10.1103/PhysRevLett.108.244101 (Cited on page 38).

Martinez, Genaro J., Andrew Adamatzky, and Harold V. McIntosh (Oct. 8, 2014). « Complete Characterization of Structure of Rule 54 ». In: *Complex Systems* 23.3. ISSN: 0891-2513 (Cited on page 103).

Mason, A, A Nicoll, and K Stratford (Jan. 1, 1991). « Synaptic Transmission between Individual Pyramidal Neurons of the Rat Visual Cortex in Vitro ». In: *The Journal of Neuroscience* 11.1, pp. 72–84. ISSN: 0270-6474, 1529-2401. DOI: 10.1523/JNEUROSCI.11-01-00072.1991 (Cited on page 35).

McDonald, Nathan (May 2017). « Reservoir Computing & Extreme Learning Machines Using Pairs of Cellular Automata Rules ». In: *2017 International Joint Conference on Neural Networks (IJCNN)*. 2017 International Joint Conference on Neural Networks (IJCNN), pp. 2429–2436. DOI: 10.1109/IJCNN.2017.7966151 (Cited on page 41).

McMullin, Barry and Francisco J. Varela (1997). « Rediscovering Computational Autopoiesis ». In: *Fourth European Conference on Artificial Life*, pp. 38–47 (Cited on page 62).

Merkel, Cory, Qutaiba Saleh, Colin Donahue, and Dhireesha Kudithipudi (Jan. 1, 2014). « Memristive Reservoir Computing Architecture for Epileptic Seizure Detection ». In: *Procedia Computer Science*. 5th Annual International Conference on Biologically Inspired Cognitive Architectures, 2014 BICA 41, pp. 249–254. ISSN: 1877-0509. DOI: 10.1016/j.procs.2014.11.110 (Cited on page 37).

Mikolov, Tomas, Armand Joulin, and Marco Baroni (2018). « A Roadmap Towards Machine Intelligence ». In: *Computational Linguistics and Intelligent Text Processing*. Ed. by Alexander Gelbukh. Lecture Notes in Computer Science. Cham: Springer International Publishing, pp. 29–61. ISBN: 978-3-319-75477-2. DOI: 10.1007/978-3-319-75477-2_2 (Cited on page 71).

Miller, Geoffrey F., Peter M. Todd, and Shailesh U. Hegde (1989). « Designing Neural Networks Using Genetic Algorithms. » In: *ICGA*. Vol. 89, pp. 379–384 (Cited on page 5).

Minsky, Marvin and Seymour A. Papert (1972). *Perceptrons: An Introduction to Computational Geometry*. 2nd edition. Cambridge MA: The MIT Press. ISBN: 0-262-63022-2. DOI: 10.7551/mitpress/11301.001.0001 (Cited on page 116).

Mitchell, Melanie (Jan. 24, 2005). « Computation in Cellular Automata: A Selected Review ». In: Gramß, Tino, Stefan Bornholdt, Michael Groß, Melanie Mitchell, and Thomas Pellizzari. *Non-Standard Computation*. Weinheim, FRG: Wiley-VCH Verlag GmbH & Co. KGaA, pp. 95–140. ISBN: 978-3-527-60296-4. DOI: 10.1002/3527602968.ch4 (Cited on page 33).

Mitchell, Melanie, James P Crutchfield, and Rajarshi Das (1996). « Evolving Cellular Automata with Genetic Algorithms: A Review of Recent Work ». In: *Proceedings of*

the First International Conference on Evolutionary Computation and Its Applications. First International Conference on Evolutionary Computation and Its Applications (EvCA '96). Moscow, Russia, p. 14 (Cited on pages 53, 64, 65, 108, 145).

Mitchell, Melanie, James P. Crutchfield, and Peter T. Hraber (Aug. 1, 1994). « Evolving Cellular Automata to Perform Computations: Mechanisms and Impediments ». In: *Physica D: Nonlinear Phenomena* 75.1, pp. 361–391. ISSN: 0167-2789. DOI: 10.1016/0167-2789(94)90293-3 (Cited on page 65).

Mitchell, Melanie, Peter Hraber, and James P. Crutchfield (Mar. 31, 1993). « Revisiting the Edge of Chaos: Evolving Cellular Automata to Perform Computations ». arXiv: adap-org/9303003 (Cited on page 65).

Morán, Alejandro, Christiam F. Frasser, and Josep L. Rosselló (June 21, 2018). « Reservoir Computing Hardware with Cellular Automata ». arXiv: 1806.04932 [nlin] (Cited on page 41).

Mordvintsev, Alexander, Ettore Randazzo, Eyvind Niklasson, and Michael Levin (Feb. 11, 2020). « Growing Neural Cellular Automata ». In: *Distill* 5.2, e23. ISSN: 2476-0757. DOI: 10.23915/distill.00023 (Cited on pages 22, 27, 28, 32, 34, 66).

Moree, Pieter and Herman J. J. te Riele (2004). « The Hexagonal versus the Square Lattice ». In: *Mathematics of Computation* 73.245, pp. 451–473. ISSN: 0025-5718. JSTOR: 4099880 (Cited on page 23).

Moreno, Matthew Andres and Charles Ofria (May 1, 2019). « Toward Open-Ended Fraternal Transitions in Individuality ». In: *Artificial Life* 25.2, pp. 117–133. ISSN: 1064-5462. DOI: 10.1162/artl_a_00284 (Cited on page 58).

Mota, Francisco, Scott Aaronson, Luís Antunes, and André Souto (2013). « Sophistication as Randomness Deficiency ». In: *Descriptional Complexity of Formal Systems*. Ed. by Helmut Jurgensen and Rogério Reis. Red. by David Hutchison et al. Vol. 8031. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 172–181. ISBN: 978-3-642-39309-9 978-3-642-39310-5. DOI: 10.1007/978-3-642-39310-5_17 (Cited on pages 51, 52).

Mouret, Jean-Baptiste and Stéphane Doncieux (2012). « Encouraging Behavioral Diversity in Evolutionary Robotics: An Empirical Study ». In: *Evolutionary Computation* 20.1, pp. 91–133. DOI: 10.1162/EVCO_a_00048 (Cited on pages 56, 57).

Müller-Schloer, Christian, ed. (2011). *Organic Computing: A Paradigm Shift for Complex Systems*. Autonomic Systems. Basel: Birkhäuser. 627 pp. ISBN: 978-3-0348-0129-4 (Cited on page 62).

Nagpal, Radhika (2001). « Programmable Self-Assembly: Constructing Global Shape Using Biologically-Inspired Local Interactions and Origami Mathematics ». PhD thesis. USA: Massachusetts Institute of Technology. 1 p. (Cited on page 69).

— (July 15, 2002). « Programmable Self-Assembly Using Biologically-Inspired Multiagent Control ». In: *Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems: Part 1*. AAMAS '02. New York, NY, USA: Association for Computing Machinery, pp. 418–425. ISBN: 978-1-58113-480-3. DOI: 10.1145/544741.544839 (Cited on pages 62, 70).

— (2008). « Programmable Pattern-Formation and Scale-Independence ». In: *Unifying Themes in Complex Systems IV*. Ed. by Ali A. Minai and Yaneer Bar-Yam. Berlin, Heidelberg: Springer, pp. 275–282. ISBN: 978-3-540-73849-7. DOI: 10.1007/978-3-540-73849-7_31 (Cited on page 62).

Nakane, Ryosho, Gouhei Tanaka, and Akira Hirose (2018). « Reservoir Computing With Spin Waves Excited in a Garnet Film ». In: *IEEE Access* 6, pp. 4462–4469. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2018.2794584 (Cited on page 38).

Nehaniv, Chrystopher L. (2003). « Evolution in Asynchronous Cellular Automata ». In: *Proceedings of the Eighth International Conference on Artificial Life*, pp. 65–73 (Cited on page 21).

Neumann, J. von (Dec. 31, 1956). « Probabilistic Logics and the Synthesis of Reliable Organisms From Unreliable Components ». In: *Automata Studies. (AM-34)*. Ed. by C. E. Shannon and J. McCarthy. Princeton University Press, pp. 43–98. ISBN: 978-1-4008-8261-8. DOI: 10.1515/9781400882618-003 (Cited on page 66).

Ng, Andrew Y., Michael I. Jordan, and Yair Weiss (Jan. 3, 2001). « On Spectral Clustering: Analysis and an Algorithm ». In: *Proceedings of the 14th International Conference on Neural Information Processing Systems: Natural and Synthetic*. NIPS'01. Cambridge, MA, USA: MIT Press, pp. 849–856 (Cited on page 116).

Nguyen, Cuong V., Yingzhen Li, Thang D. Bui, and Richard E. Turner (2017). « Variational Continual Learning ». In: *CoRR* abs/1710.10628. arXiv: 1710.10628 (Cited on page 117).

Nichele, Stefano and Magnus S. Gundersen (Feb. 13, 2017). « Reservoir Computing Using Non-Uniform Binary Cellular Automata ». arXiv: 1702.03812 [cs] (Cited on pages 41, 127).

Nichele, Stefano and Andreas Molund (2017a). « Deep Learning with Cellular Automaton-Based Reservoir Computing ». In: *0891-2513*. ISSN: 0891-2513. DOI: http://dx.doi.org/10.25088/ComplexSystems.26.4.319 (Cited on pages 41, 127).

— (Mar. 8, 2017b). « Deep Reservoir Computing Using Cellular Automata ». arXiv: 1703.02806 [cs] (Cited on page 41).

Niklasson, Eyvind, Alexander Mordvintsev, Ettore Randazzo, and Michael Levin (Feb. 11, 2021). « Self-Organising Textures ». In: *Distill* 6.2, e00027–003. DOI: 10.23915/distill.00027.003 (Cited on page 67).

Nishio, Hidenosuke and Youichi Kobuchi (Oct. 1, 1975). « Fault Tolerant Cellular Spaces ». In: *Journal of Computer and System Sciences* 11.2, pp. 150–170. ISSN: 0022-0000. DOI: 10.1016/S0022-0000(75)80065-1 (Cited on page 66).

Ofria, Charles and Claus O. Wilke (2004). « Avida: A Software Platform for Research in Computational Evolutionary Biology ». In: *Artificial Life* 10.2, pp. 191–229. ISSN: 1064-5462. DOI: 10.1162/106454604773563612. pmid: 15107231 (Cited on pages 60, 72).

Ostrovsky, B., G. Crooks, M. A. Smith, and Y. Bar-Yam (Apr. 1, 2001). « Cellular Automata for Polymer Simulation with Application to Polymer Melts and Polymer Collapse Including Implications for Protein Folding ». In: *Parallel Computing*. Cel-

lular Automata: From Modeling to Applications 27.5, pp. 613–641. ISSN: 0167-8191. DOI: 10.1016/S0167-8191(00)00081-8 (Cited on page 61).

Ottavi, H. and O. Parodi (Apr. 1989). « Simulation of the Ising Model by Cellular Automata ». In: *Europhysics Letters (EPL)* 8.8, pp. 741–746. ISSN: 0295-5075. DOI: 10.1209/0295-5075/8/8/006 (Cited on page 21).

Packard, Norman, Mark A. Bedau, Alastair Channon, Takashi Ikegami, Steen Rasmussen, Kenneth O. Stanley, and Tim Taylor (May 1, 2019). « An Overview of Open-Ended Evolution: Editorial Introduction to the Open-Ended Evolution II Special Issue ». In: *Artificial Life* 25.2, pp. 93–103. ISSN: 1064-5462. DOI: 10.1162/artl_a_00291 (Cited on page 57).

Packard, Norman H and Stephen Wolfram (1985). « Two-Dimensional Cellular Automata ». In: *Journal of Statistical Physics* 38.5/6, p. 46 (Cited on page 73).

Packard, Norman H. (1988). « Adaptation toward the Edge of Chaos ». In: *Dynamic patterns in complex systems* 212, pp. 293–301 (Cited on page 64).

Packel, Edward W. and Henryk Woźniakowski (1987). « Recent Developments in Information-Based Complexity ». In: *Bulletin of the American Mathematical Society* 17.1, pp. 9–36. ISSN: 0273-0979, 1088-9485. DOI: 10.1090/S0273-0979-1987-15511-X (Cited on page 47).

Pan, Peng-Zhi, Xia-Ting Feng, and John A. Hudson (2009). « Study of Failure and Scale Effects in Rocks under Uniaxial Compression Using 3D Cellular Automata ». In: *International Journal of Rock Mechanics and Mining Sciences* 46.4, pp. 674–685 (Cited on page 22).

Paquot, Y., F. Duport, A. Smerieri, J. Dambre, B. Schrauwen, M. Haelterman, and S. Massar (Feb. 27, 2012). « Optoelectronic Reservoir Computing ». In: *Scientific Reports* 2.1 (1), p. 287. ISSN: 2045-2322. DOI: 10.1038/srep00287 (Cited on page 38).

Parisi, German I., Ronald Kemker, Jose L. Part, Christopher Kanan, and Stefan Wermter (May 2019). « Continual Lifelong Learning with Neural Networks: A Review ». In: *Neural Networks* 113, pp. 54–71. ISSN: 08936080. DOI: 10.1016/j.neunet.2019.01.012.. arXiv: 1802.07569 (Cited on page 4).

Pascanu, Razvan and Herbert Jaeger (Mar. 1, 2011). « A Neurodynamical Model for Working Memory ». In: *Neural Networks* 24.2, pp. 199–207. ISSN: 0893-6080. DOI: 10.1016/j.neunet.2010.10.003 (Cited on page 37).

Pathak, Deepak, Chris Lu, Trevor Darrell, Phillip Isola, and Alexei A Efros (2019). « Learning to Control Self-Assembling Morphologies: A Study of Generalization via Modularity ». In: *NeurIPS*. NeurIPS, p. 13 (Cited on page 72).

Pattee, Howard H. and Hiroki Sayama (Apr. 2019). « Evolved Open-Endedness, Not Open-Ended Evolution ». In: *Artificial Life* 25.1, pp. 4–8. ISSN: 1064-5462, 1530-9185. DOI: 10.1162/artl_a_00276 (Cited on page 58).

Pesavento, Umberto (July 1, 1995). « An Implementation of von Neumann's Self-Reproducing Machine ». In: *Artificial Life* 2.4, pp. 337–354. ISSN: 1064-5462. DOI: 10.1162/artl.1995.2.4.337 (Cited on page 60).

Pfeifer, Rolf, Josh Bongard, and Simon Grand (2007). *How the Body Shapes the Way We Think: A New View of Intelligence*. Cambridge, Mass: MIT Press. 394 pp. ISBN: 978-0-262-16239-5 (Cited on page 37).

Pivato, Marcus (Feb. 22, 2007). « Spectral Domain Boundaries in Cellular Automata ». arXiv: math/0507091 (Cited on page 103).

Poli, Riccardo and Brian Logan (1996). « On the Relations between Search and Evolutionary Algorithms ». In: *COGNITIVE SCIENCE RESEARCH PAPERS-UNIVERSITY OF BIRMINGHAM CSRP* (Cited on page 53).

Potoyan, Davit A., Alexey Savelyev, and Garegin A. Papoian (Jan. 2013). « Recent Successes in Coarse-Grained Modeling of DNA: Coarse-grained Modeling of DNA ». In: *Wiley Interdisciplinary Reviews: Computational Molecular Science* 3.1, pp. 69–83. ISSN: 17590876. DOI: 10.1002/wcms.1114 (Cited on page 94).

Pugh, Justin K., Lisa B. Soros, and Kenneth O. Stanley (2016). « Quality Diversity: A New Frontier for Evolutionary Computation ». In: *Frontiers in Robotics and AI* 3. ISSN: 2296-9144. DOI: 10.3389/frobt.2016.00040 (Cited on page 56).

Rafayelyan, Mushegh, Jonathan Dong, Yongqi Tan, Florent Krzakala, and Sylvain Gigan (Nov. 20, 2020). « Large-Scale Optical Reservoir Computing for Spatiotemporal

Chaotic Systems Prediction ». In: *Physical Review X* 10.4, p. 041037. DOI: 10.1103/PhysRevX.10.041037 (Cited on page 38).

Randazzo, Ettore, Alexander Mordvintsev, Eyvind Niklasson, Michael Levin, and Sam Greydanus (Aug. 27, 2020). « Self-Classifying MNIST Digits ». In: *Distill* 5.8, e00027.002. ISSN: 2476-0757. DOI: 10.23915/distill.00027.002 (Cited on pages 32, 67).

Ray, Thomas S. (Jan. 1, 1991). « An Approach to the Synthesis of Life ». In: *Artificial Life II*. 1st ed. C. G. Langton et al. (Eds.) Addison-Wesley Publishing Co, pp. 371–408 (Cited on pages 58, 60, 72).

Rebuffi, Sylvestre-Alvise, Alexander Kolesnikov, Georg Sperl, and Christoph H. Lampert (2017). « Icarl: Incremental Classifier and Representation Learning ». In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2001–2010 (Cited on page 137).

Rényi, Alfréd (Jan. 1, 1961). « On Measures of Entropy and Information ». In: *Proceedings of the Fourth Berkeley Symposium on Mathematical Statistics and Probability, Volume 1: Contributions to the Theory of Statistics* 4.1, pp. 547–562 (Cited on page 46).

Richards, Fred C., Thomas P. Meyer, and Norman H. Packard (Sept. 2, 1990). « Extracting Cellular Automaton Rules Directly from Experimental Data ». In: *Physica D: Nonlinear Phenomena* 45.1, pp. 189–202. ISSN: 0167-2789. DOI: 10.1016/0167-2789(90)90182-O (Cited on page 64).

Richardson, Kyle, Hai Hu, Lawrence Moss, and Ashish Sabharwal (2020). « Probing Natural Language Inference Models through Semantic Fragments ». In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 34. 05, pp. 8713–8721 (Cited on page 117).

Risi, Sebastian, Charles E. Hughes, and Kenneth O. Stanley (2010). « Evolving Plastic Neural Networks with Novelty Search ». In: *Adaptive Behavior* 18.6, pp. 470–491 (Cited on page 55).

Robins, Anthony (1993). « Catastrophic Forgetting in Neural Networks: The Role of Rehearsal Mechanisms ». In: *Proceedings 1993 The First New Zealand International*

Two-Stream Conference on Artificial Neural Networks and Expert Systems. IEEE, pp. 65–68 (Cited on page 137).

Rumelhart, David E., Geoffrey E. Hinton, and Ronald J. Williams (1985). *Learning Internal Representations by Error Propagation*. California Univ San Diego La Jolla Inst for Cognitive Science (Cited on page 116).

Rupe, Adam and James P. Crutchfield (July 2018). « Local Causal States and Discrete Coherent Structures ». In: *Chaos: An Interdisciplinary Journal of Nonlinear Science* 28.7, p. 075312. ISSN: 1054-1500, 1089-7682. DOI: 10.1063/1.5021130. arXiv: 1801.00515 (Cited on page 103).

Ryabov, Vladimir and Dmitry Nerukh (Sept. 2011). « Computational Mechanics of Molecular Systems: Quantifying High-Dimensional Dynamics by Distribution of Poincaré Recurrence Times ». In: *Chaos: An Interdisciplinary Journal of Nonlinear Science* 21.3, p. 037113. ISSN: 1054-1500. DOI: 10.1063/1.3608125 (Cited on page 51).

Salzberg, Chris and Hiroki Sayama (2004). « Complex Genetic Evolution of Artificial Self-Replicators in Cellular Automata ». In: *Complex.* 10.2, pp. 33–39. DOI: 10.1002/cplx.20060 (Cited on page 60).

San Miguel, M., J. H. Johnson, J. Kertesz, K. Kaski, A. Díaz-Guilera, R. S. MacKay, V. Loreto, P. Érdi, and D. Helbing (Nov. 1, 2012). « Challenges in Complex Systems Science ». In: *The European Physical Journal Special Topics* 214.1, pp. 245–271. ISSN: 1951-6401. DOI: 10.1140/epjst/e2012-01694-y (Cited on page 6).

Sapin, Emmanuel, Olivier Bailleux, and Chabrier Jean-Jacques (2003). « Research of a Cellular Automaton Simulating Logic Gates by Evolutionary Algorithms ». In: *Genetic Programming*. Ed. by Conor Ryan, Terence Soule, Maarten Keijzer, Edward Tsang, Riccardo Poli, and Ernesto Costa. Red. by G. Goos, J. Hartmanis, and J. van Leeuwen. Vol. 2610. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 414–423. ISBN: 978-3-540-00971-9 978-3-540-36599-0. DOI: 10.1007/3-540-36599-0_39 (Cited on page 108).

Sayama, Hiroki (1999). « A New Structurally Dissolvable Self-Reproducing Loop Evolving in a Simple Cellular Automata Space ». In: *Artificial Life* 5.4, pp. 343–365. ISSN: 1064-5462. DOI: 10.1162/106454699568818. pmid: 10829086 (Cited on page 60).

— (Apr. 2011). « Seeking Open-Ended Evolution in Swarm Chemistry ». In: *2011 IEEE Symposium on Artificial Life (ALIFE)*. 2011 Ieee Symposium On Artificial Life - Part Of 17273 - 2011 Ssci. Paris, France: IEEE, pp. 186–193. ISBN: 978-1-61284-062-8. DOI: 10.1109/ALIFE.2011.5954667 (Cited on pages 60, 61).

— (May 1, 2019). « Cardinality Leap for Open-Ended Evolution: Theoretical Consideration and Demonstration by Hash Chemistry ». In: *Artificial Life* 25.2, pp. 104–116. ISSN: 1064-5462. DOI: 10.1162/artl_a_00283 (Cited on page 58).

Schmidhuber, Jürgen (2002). « The Speed Prior: A New Simplicity Measure Yielding Near-Optimal Computable Predictions ». In: *Computational Learning Theory*. Ed. by Jyrki Kivinen and Robert H. Sloan. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, pp. 216–228. ISBN: 978-3-540-45435-9. DOI: 10.1007/3-540-45435-7_15 (Cited on page 48).

Schmidhuber, Jürgen and S. Heil (Jan. 1996). « Sequential Neural Text Compression ». In: *IEEE Transactions on Neural Networks* 7.1, pp. 142–146. ISSN: 1045-9227. DOI: 10.1109/72.478398 (Cited on page 80).

Schmidhuber, Jürgen, Daan Wierstra, Matteo Gagliolo, and Faustino Gomez (Mar. 1, 2007). « Training Recurrent Networks by Evolino ». In: *Neural Computation* 19.3, pp. 757–779. ISSN: 0899-7667. DOI: 10.1162/neco.2007.19.3.757 (Cited on page 37).

Schomaker, Lambert R. B. (1990). « Neural Network Models of Temporal Pattern Generation. » In: Conference on Sequencing and Timing of Human Movement. Wassenaar (The Netherlands): NIAS. Wassenaar, The Netherlands, p. 27 (Cited on page 35).

— (1991). « Simulation and Recognition of Handwriting Movements ». Nijmegen, Netherlands: Radboud University. 223 pp. (Cited on page 35).

— (Feb. 1, 1992). « A Neural Oscillator-Network Model of Temporal Pattern Generation ». In: *Human Movement Science* 11.1, pp. 181–192. ISSN: 0167-9457. DOI: 10.1016/0167-9457(92)90059-K (Cited on page 35).

Schönfinkel, M. (Sept. 1, 1924). « Über die Bausteine der mathematischen Logik [About the building blocks of mathematical logic] ». In: *Mathematische Annalen* 92.3, pp. 305–316. ISSN: 1432-1807. DOI: 10.1007/BF01448013 (Cited on page 61).

Secretan, Jimmy, Nicholas Beato, David B. D'Ambrosio, Adelein Rodriguez, Adam Campbell, Jeremiah T. Folsom-Kovarik, and Kenneth O. Stanley (2011). « Picbreeder: A Case Study in Collaborative Evolutionary Exploration of Design Space ». In: *Evolutionary Computation* 19.3, pp. 373–403. ISSN: 1530-9304. DOI: 10.1162/EVCO_a_00030. pmid: 20964537 (Cited on page 68).

Shalizi, Cosma Rohilla and James P. Crutchfield (Aug. 1, 2001). « Computational Mechanics: Pattern and Prediction, Structure and Simplicity ». In: *Journal of Statistical Physics* 104.3, pp. 817–879. ISSN: 1572-9613. DOI: 10.1023/A:1010388907793 (Cited on page 51).

Shalizi, Cosma Rohilla, Robert Haslinger, Jean-Baptiste Rouquier, Kristina Lisa Klinkner, and Cristopher Moore (Mar. 2, 2006). « Automatic Filters for the Detection of Coherent Structure in Spatiotemporal Systems ». In: *Physical Review E* 73.3. ISSN: 1539-3755, 1550-2376. DOI: 10.1103/PhysRevE.73.036104 (Cited on page 95).

Shalizi, Cosma Rohilla, Kristina Lisa Shalizi, and Robert Haslinger (Sept. 10, 2004). « Quantifying Self-Organization with Optimal Predictors ». In: *Physical Review Letters* 93.11, p. 118701. ISSN: 0031-9007, 1079-7114. DOI: 10.1103/PhysRevLett.93.118701. arXiv: nlin/0409024 (Cited on pages 51, 95).

Shannon, Claude E. and Warren Weaver (1975). *The Mathematical Theory of Communication*. Urbana: University of Illinois Press. 125 pp. ISBN: 978-0-252-72548-7 (Cited on pages 45, 49).

Shaw, Robert (1984). *The Dripping Faucet as a Model Chaotic System*. The Science Frontier Express Series. Santa Cruz, CA: Aerial Press. 111 pp. ISBN: 978-0-942344-05-9 (Cited on page 46).

Siegelmann, Hava T. and Eduardo D. Sontag (1992). « On the Computational Power of Neural Nets ». In: *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, pp. 440–449 (Cited on page 34).

Simon, Herbert A. (1962). « The Architecture of Complexity ». In: *Proceedings of the American Philosophical Society* 106.6, pp. 467–482. ISSN: 0003-049X. JSTOR: 985254 (Cited on page 109).

Sims, Karl (1994). « Evolving Virtual Creatures ». In: *SIGGRAPH*. DOI: 10.1145/192161.192167 (Cited on pages 60, 72).

Sipper, Moshe (May 1, 1996). « Co-Evolving Non-Uniform Cellular Automata to Perform Computations ». In: *Physica D: Nonlinear Phenomena* 92.3, pp. 193–208. ISSN: 0167-2789. DOI: 10.1016/0167-2789(95)00286-3 (Cited on page 65).

Smith, Alvy Ray (July 1, 1971). « Simple Computation-Universal Cellular Spaces ». In: *Journal of the ACM* 18.3, pp. 339–353. ISSN: 0004-5411. DOI: 10.1145/321650.321652 (Cited on page 63).

Smith, John Maynard and Eors Szathmary (Mar. 16, 2000). *The Origins of Life: From the Birth of Life to the Origin of Language*. OUP Oxford. 191 pp. ISBN: 978-0-19-164732-1 (Cited on page 5).

Soh, Harold and Yiannis Demiris (June 2012). « Iterative Temporal Learning and Prediction with the Sparse Online Echo State Gaussian Process ». In: *The 2012 International Joint Conference on Neural Networks (IJCNN)*. The 2012 International Joint Conference on Neural Networks (IJCNN), pp. 1–8. DOI: 10.1109/IJCNN.2012.6252504 (Cited on page 38).

Soler-Toscano, Fernando, Hector Zenil, Jean-Paul Delahaye, and Nicolas Gauvrit (May 8, 2014). « Calculating Kolmogorov Complexity from the Output Frequency Distributions of Small Turing Machines ». In: *PLOS ONE* 9.5, e96223. ISSN: 1932-6203. DOI: 10.1371/journal.pone.0096223 (Cited on pages 75, 93).

Solomonoff, R.J. (Mar. 1964a). « A Formal Theory of Inductive Inference. Part I ». In: *Information and Control* 7.1, pp. 1–22. ISSN: 00199958. DOI: 10.1016/S0019-9958(64)90223-2 (Cited on page 47).

— (June 1964b). « A Formal Theory of Inductive Inference. Part II ». In: *Information and Control* 7.2, pp. 224–254. ISSN: 00199958. DOI: 10.1016/S0019-9958(64)90131-7 (Cited on page 47).

Solomonoff, Ray J. (1960). « A Preliminary Report on a General Theory of Inductive Inference ». In: Zator Company Cambridge, MA (Cited on page 47).

Soltoggio, Andrea, John A Bullinaria, Claudio Mattiussi, Peter Durr, and Dario Floreano (2008). « Evolutionary Advantages of Neuromodulated Plasticity in Dynamic, Reward-based Scenarios ». In: p. 8 (Cited on page 32).

Soros, Lisa B. and Kenneth O. Stanley (July 30, 2014). « Identifying Necessary Conditions for Open-Ended Evolution through the Artificial Life World of Chromaria ». In: *Artificial Life 14: Proceedings of the Fourteenth International Conference on the Synthesis and Simulation of Living Systems*. Artificial Life 14: International Conference on the Synthesis and Simulation of Living Systems. The MIT Press, pp. 793–800. ISBN: 978-0-262-32621-6. DOI: 10.7551/978-0-262-32621-6-ch128 (Cited on pages 59, 60, 72).

Soros, Lisa B., Kenneth O. Stanley, and Joel Lehman (Dec. 19, 2017). *Open-Endedness: The Last Grand Challenge You've Never Heard Of*. O'Reilly Media. URL: https://www.oreilly.com/radar/open-endedness-the-last-grand-challenge-youve-never-heard-of/ (visited on 05/10/2021) (Cited on page 57).

Spector, Lee, Jon Klein, and Mark Feinstein (2007). « Division Blocks and the Open-ended Evolution of Development, Form, and Behavior ». In: *Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation* (London, England). GECCO '07. New York, NY, USA: ACM, pp. 316–323. ISBN: 978-1-59593-697-4. DOI: 10.1145/1276958.1277019 (Cited on pages 60, 72).

Srivastava, Rupesh Kumar, Jonathan Masci, Sohrob Kazerounian, Faustino J. Gomez, and Jürgen Schmidhuber (2013). « Compete to Compute ». In: *Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems 2013. Proceedings of a Meeting Held December 5-8, 2013, Lake Tahoe, Nevada, United States*. Ed. by Christopher J. C. Burges, Léon Bottou, Zoubin Ghahramani, and Kilian Q. Weinberger, pp. 2310–2318 (Cited on page 117).

Stanley, Kenneth O. (Aug. 1, 2019). « Why Open-Endedness Matters ». In: *Artificial Life* 25.3, pp. 232–235. ISSN: 1064-5462. DOI: 10.1162/artl_a_00294 (Cited on page 57).

Stanley, Kenneth O. and Joel Lehman (2015). *Why Greatness Cannot Be Planned: The Myth of the Objective*. Cham, Switzerland: Springer International Publishing. 141 pp. ISBN: 978-3-319-15523-4 (Cited on page 6).

Stanley, Kenneth O. and Risto Miikkulainen (June 2002). « Evolving Neural Networks through Augmenting Topologies ». In: *Evolutionary Computation* 10.2, pp. 99–127. ISSN: 1063-6560, 1530-9304. DOI: 10.1162/106365602320169811 (Cited on page 57).

Sudhakaran, Shyam, Djordje Grbic, Siyan Li, Adam Katona, Elias Najarro, Claire Glanois, and Sebastian Risi (June 4, 2021). « Growing 3D Artefacts and Functional Machines with Neural Cellular Automata ». arXiv: 2103.08737 [cs] (Cited on page 22).

Sugano, Chihiro, Kazutaka Kanno, and Atsushi Uchida (Jan. 2020). « Reservoir Computing Using Multiple Lasers With Feedback on a Photonic Integrated Circuit ». In: *IEEE Journal of Selected Topics in Quantum Electronics* 26.1, pp. 1–9. ISSN: 1558-4542. DOI: 10.1109/JSTQE.2019.2929179 (Cited on page 38).

Tanaka, Gouhei, Toshiyuki Yamane, Jean Benoit Héroux, Ryosho Nakane, Naoki Kanazawa, Seiji Takeda, Hidetoshi Numata, Daiju Nakano, and Akira Hirose (July 1, 2019). « Recent Advances in Physical Reservoir Computing: A Review ». In: *Neural Networks* 115, pp. 100–123. ISSN: 0893-6080. DOI: 10.1016/j.neunet.2019.03.005 (Cited on pages 35, 36, 38).

Taylor, Matthew E. and Peter Stone (2007). « Cross-Domain Transfer for Reinforcement Learning ». In: *Machine Learning, Proceedings of the Twenty-Fourth International Conference (ICML 2007), Corvallis, Oregon, USA, June 20-24, 2007*. Ed. by Zoubin Ghahramani. Vol. 227. ACM International Conference Proceeding Series. ACM, pp. 879–886. DOI: 10.1145/1273496.1273607 (Cited on page 115).

Taylor, Matthew E., Peter Stone, and Yaxin Liu (2007). « Transfer Learning via Inter-Task Mappings for Temporal Difference Learning ». In: *J. Mach. Learn. Res.* 8, pp. 2125–2167 (Cited on page 115).

Taylor, Tim (Jan. 1, 2002). « Chapter 1 - Creativity in Evolution: Individuals, Interactions, and Environments ». In: *Creative Evolutionary Systems*. Ed. by Peter J Bentley and David W. Corne. The Morgan Kaufmann Series in Artificial Intelli-

gence. San Francisco: Morgan Kaufmann, pp. 79–108. ISBN: 978-1-55860-673-9. DOI: 10.1016/B978-155860673-9/50037-9 (Cited on page 60).

Taylor, Tim (July 27, 2015). *Requirements for Open-Ended Evolution in Natural and Artificial Systems*. DOI: 10.48550/arXiv.1507.07403. arXiv: 1507.07403 [cs, q-bio]. URL: http://arxiv.org/abs/1507.07403 (visited on 11/09/2022). preprint (Cited on page 59).

Taylor, Tim et al. (Aug. 2016). « Open-Ended Evolution: Perspectives from the OEE Workshop in York ». In: *Artificial Life* 22.3, pp. 408–423. ISSN: 1064-5462. DOI: 10.1162/ARTL_a_00210 (Cited on pages 57, 59).

Torrejon, Jacob et al. (July 2017). « Neuromorphic Computing with Nanoscale Spintronic Oscillators ». In: *Nature* 547.7664 (7664), pp. 428–431. ISSN: 1476-4687. DOI: 10.1038/nature23011 (Cited on page 38).

Traub, J. F., G. W. Wasilkowski, and H. Woźniakowski (1983). *Information, Uncertainty, Complexity*. Reading, Mass: Addison-Wesley Pub. Co., Advanced Book Program/World Science Division. 176 pp. ISBN: 978-0-201-07890-9 (Cited on page 47).

Tsalides, Ph, P. J. Hicks, and T. A. York (Nov. 1, 1989). « Three-Dimensional Cellular Automata and VLSI Applications ». In: *IEE Proceedings E (Computers and Digital Techniques)* 136.6, pp. 490–495. ISSN: 2053-7948. DOI: 10.1049/ip-e.1989.0067 (Cited on page 22).

Tsertsvadze, G. N. (1964). « Stochastic Automata and the Problem of Constructing Reliable Automata from Unreliable Elements. i(Reliable Finite Automata Design from Unreliable Elements, Using Stochastic Automaton Model) ». In: *Automation and Remote Control* 25, pp. 198–210 (Cited on page 66).

Tyson, J. J. (Mar. 13, 2013). *The Belousov-Zhabotinskii Reaction*. Springer Science & Business Media. 138 pp. ISBN: 978-3-642-93046-1. Google Books: hIPqCAAAQBAJ (Cited on page 53).

Vandoorne, Kristof, Joni Dambre, David Verstraeten, Benjamin Schrauwen, and Peter Bienstman (Sept. 2011). « Parallel Reservoir Computing Using Optical Amplifiers ».

In: *IEEE Transactions on Neural Networks* 22.9, pp. 1469–1481. ISSN: 1941-0093. DOI: 10.1109/TNN.2011.2161771 (Cited on page 38).

Vandoorne, Kristof, Wouter Dierckx, Benjamin Schrauwen, David Verstraeten, Roel Baets, Peter Bienstman, and Jan Van Campenhout (July 21, 2008). « Toward Optical Signal Processing Using Photonic Reservoir Computing ». In: *Optics Express* 16.15, pp. 11182–11192. ISSN: 1094-4087. DOI: 10.1364/OE.16.011182 (Cited on page 38).

Vandoorne, Kristof, Pauline Mechet, Thomas Van Vaerenbergh, Martin Fiers, Geert Morthier, David Verstraeten, Benjamin Schrauwen, Joni Dambre, and Peter Bienstman (Mar. 24, 2014). « Experimental Demonstration of Reservoir Computing on a Silicon Photonics Chip ». In: *Nature Communications* 5.1 (1), p. 3541. ISSN: 2041-1723. DOI: 10.1038/ncomms4541 (Cited on page 38).

Varela, Francisco G., Humberto R. Maturana, and Ricardo Uribe (1991). « Autopoiesis: The Organization of Living Systems, Its Characterization and a Model ». In: *Facets of Systems Science*. Springer, pp. 559–569 (Cited on page 62).

Vaswani, Ashish, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin (Dec. 5, 2017). « Attention Is All You Need ». arXiv: 1706.03762 [cs] (Cited on pages 114, 124, 130, 135).

Verstraeten, D., B. Schrauwen, D. Stroobandt, and J. Van Campenhout (Sept. 30, 2005). « Isolated Word Recognition with the Liquid State Machine: A Case Study ». In: *Information Processing Letters*. Applications of Spiking Neural Networks 95.6, pp. 521–528. ISSN: 0020-0190. DOI: 10.1016/j.ipl.2005.05.019 (Cited on page 38).

Verstraeten, David, Benjamin Schrauwen, and Dirk Stroobandt (2005). « Reservoir Computing with Stochastic Bitstream Neurons ». In: *Proceedings of the 16th Annual ProRISC Workshop*, pp. 454–459 (Cited on page 37).

Vichniac, Gérard Y. (Jan. 1, 1984). « Simulating Physics with Cellular Automata ». In: *Physica D: Nonlinear Phenomena* 10.1, pp. 96–116. ISSN: 0167-2789. DOI: 10.1016/0167-2789(84)90253-7 (Cited on page 21).

Vispoel, Milan, Aisling J. Daly, and Jan M. Baetens (Apr. 1, 2022). « Progress, Gaps and Obstacles in the Classification of Cellular Automata ». In: *Physica D: Nonlinear*

Phenomena 432, p. 133074. ISSN: 0167-2789. DOI: 10.1016/j.physd.2021.133074 (Cited on page 18).

Vitanyi, P.M. (Oct. 2006). « Meaningful Information ». In: *IEEE Transactions on Information Theory* 52.10, pp. 4617–4626. ISSN: 1557-9654. DOI: 10.1109/TIT.2006.881729 (Cited on page 51).

Von Neumann, John and Arthur W. Burks (1966). « Theory of Self-Reproducing Automata ». In: *IEEE Transactions on Neural Networks* 5.1, pp. 3–14 (Cited on pages 4, 13, 60, 63, 73, 92).

Wang, Yaqing, Quanming Yao, James T. Kwok, and Lionel M. Ni (2020). « Generalizing from a Few Examples: A Survey on Few-Shot Learning ». In: *ACM computing surveys (csur)* 53.3, pp. 1–34 (Cited on page 116).

Weiss, Ron, George Homsy, and Radhika Nagpal (1998). « Programming Biological Cells ». In: *Proceedings of the 8th International Conference on Architectural Support for Programming Languages and Operating Systems*. International Conference on Architectural Support for Programming Languages and Operating Systems, p. 1 (Cited on page 69).

Weiss, Ron, George E. Homsy, and Thomas F. Knight (2002). « Toward in Vivo Digital Circuits ». In: *Evolution as Computation*. Ed. by Laura F. Landweber and Erik Winfree. Red. by G. Rozenberg, Th. Bäck, A. E. Eiben, J. N. Kok, and H. P. Spaink. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 275–295. ISBN: 978-3-642-63081-1 978-3-642-55606-7. DOI: 10.1007/978-3-642-55606-7_14 (Cited on page 69).

Weston, Jason, Antoine Bordes, Sumit Chopra, and Tomas Mikolov (2016). « Towards AI-Complete Question Answering: A Set of Prerequisite Toy Tasks ». In: *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*. Ed. by Yoshua Bengio and Yann LeCun (Cited on page 117).

Williame, Jérôme, Artur Difini Accioly, Damien Rontani, Marc Sciamanna, and Joo-Von Kim (June 10, 2019). « Chaotic Dynamics in a Macrospin Spin-Torque Nano-Oscillator with Delayed Feedback ». In: *Applied Physics Letters* 114.23, p. 232405. ISSN: 0003-6951. DOI: 10.1063/1.5095630 (Cited on page 38).

Winograd, Shmuel and Jack D. Cowan (1963). *Reliable Computation in the Presence of Noise*. Mit Press Cambridge, Mass. (Cited on page 66).

Woese, Carl (June 9, 1998). « The Universal Ancestor ». In: *Proceedings of the National Academy of Sciences* 95.12, pp. 6854–6859. DOI: 10.1073/pnas.95.12.6854 (Cited on page 5).

Wolf, S. A., D. D. Awschalom, R. A. Buhrman, J. M. Daughton, S. von Molnár, M. L. Roukes, A. Y. Chtchelkanova, and D. M. Treger (Nov. 16, 2001). « Spintronics: A Spin-Based Electronics Vision for the Future ». In: *Science* 294.5546, pp. 1488–1495. DOI: 10.1126/science.1065389 (Cited on page 38).

Wolfram, Stephen (July 1, 1983). « Statistical Mechanics of Cellular Automata ». In: *Reviews of Modern Physics* 55.3, pp. 601–644. ISSN: 0034-6861. DOI: 10.1103/RevModPhys.55.601 (Cited on pages 20, 45, 46, 126).

— (Jan. 1, 1984). « Universality and Complexity in Cellular Automata ». In: *Physica D: Nonlinear Phenomena* 10.1, pp. 1–35. ISSN: 0167-2789. DOI: 10.1016/0167-2789(84)90245-8 (Cited on pages 16, 17, 73).

— (2002). *A New Kind of Science*. Champaign, IL: Wolfram Media. 1197 pp. ISBN: 978-1-57955-008-0 (Cited on pages 8, 19, 96, 108, 126).

Wolpert, L. (Oct. 1, 1969). « Positional Information and the Spatial Pattern of Cellular Differentiation ». In: *Journal of Theoretical Biology* 25.1, pp. 1–47. ISSN: 0022-5193. DOI: 10.1016/S0022-5193(69)80016-0 (Cited on page 70).

Woolley, Brian G. and Kenneth O. Stanley (July 12, 2011). « On the Deleterious Effects of a Priori Objectives on Evolution and Representation ». In: *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation*. GECCO '11. New York, NY, USA: Association for Computing Machinery, pp. 957–964. ISBN: 978-1-4503-0557-0. DOI: 10.1145/2001576.2001707 (Cited on page 68).

Wuensche, Andrew (1999). « Classifying Cellular Automata Automatically: Finding Gliders, Filtering, and Relating Space-Time Patterns, Attractor Basins, and the Z Parameter ». In: *Complexity* 4.3, pp. 47–66. ISSN: 1099-0526. DOI: 10.1002/(SICI)

1099-0526(199901/02)4:3<47::AID-CPLX9>3.0.CO;2-V (Cited on pages 19, 95, 102).

Wuensche, Andrew (2011). *Exploring Discrete Dynamics*. Luniver Press (Cited on page 102).

Wuensche, Andrew and Mike Lesser (1992). *The Global Dynamics of Cellular Automata: An Atlas of Basin of Attraction Fields of One-Dimensional Cellular Automata*. Santa Fe Institute Studies in the Sciences of Complexity. Reference Volume v. 1. Reading, Mass: Addison-Wesley. 250 pp. ISBN: 978-0-201-55740-4 (Cited on pages 18, 19).

Wulff, N. H. and J. A. Hertz (1993). « Learning Cellular Automaton Dynamics with Neural Networks ». In: *Advances in Neural Information Processing Systems*, pp. 631–638 (Cited on page 27).

Wyffels, F. and B. Schrauwen (June 1, 2010). « A Comparative Study of Reservoir Computing Strategies for Monthly Time Series Prediction ». In: *Neurocomputing*. Subspace Learning / Selected Papers from the European Symposium on Time Series Prediction 73.10, pp. 1958–1964. ISSN: 0925-2312. DOI: 10.1016/j.neucom.2010.01.016 (Cited on page 38).

Yaeger, Larry (1994). « Computational Genetics, Physiology, Metabolism, Neural Systems, Learning, Vision, and Behavior or Poly World: Life in a New Context ». In: *Santa Fe Institute Studies in the Sciences of Complexity*. Vol. 17. Addison-Wesley Publishing Co, pp. 263–263 (Cited on pages 60, 72).

Yamazaki, Tadashi and Shigeru Tanaka (Apr. 2007). « The Cerebellum as a Liquid State Machine ». In: *Neural Networks: The Official Journal of the International Neural Network Society* 20.3, pp. 290–297. ISSN: 0893-6080. DOI: 10.1016/j.neunet.2007.04.004. pmid: 17517494 (Cited on page 37).

Yang, Xiao, Wanlong Chen, and Frank Z. Wang (May 1, 2016). « Investigations of the Staircase Memristor Model and Applications of Memristor-Based Local Connections ». In: *Analog Integrated Circuits and Signal Processing* 87.2, pp. 263–273. ISSN: 1573-1979. DOI: 10.1007/s10470-016-0715-3 (Cited on page 37).

Yarats, Denis, Amy Zhang, Ilya Kostrikov, Brandon Amos, Joelle Pineau, and Rob Fergus (2019). « Improving Sample Efficiency in Model-Free Reinforcement Learning from Images ». arXiv: 1910.01741 (Cited on page 116).

Yildiz, Izzet B., Herbert Jaeger, and Stefan J. Kiebel (Nov. 1, 2012). « Re-Visiting the Echo State Property ». In: *Neural Networks* 35, pp. 1–9. ISSN: 0893-6080. DOI: 10.1016/j.neunet.2012.07.005 (Cited on page 36).

Yilmaz, Ozgür (Oct. 1, 2014). « Reservoir Computing Using Cellular Automata ». arXiv: 1410.0162 [cs] (Cited on pages 40, 41, 115, 127, 128, 135).

Zeleny, Milan (1977). « Self-Organization of Living Systems: A Formal Model of Autopoiesis ». In: *International journal of general system* 4.1, pp. 13–28 (Cited on page 62).

Zenil, Hector (2010). « Compression-Based Investigation of the Dynamical Properties of Cellular Automata and Other Systems ». In: *Complex Systems* 19.1 (Cited on pages 17, 19, 48, 74, 76, 85, 86, 93).

— (Sept. 1, 2014). « What Is Nature-Like Computation? A Behavioural Approach and a Notion of Programmability ». In: *Philosophy & Technology* 27.3, pp. 399–421. ISSN: 2210-5441. DOI: 10.1007/s13347-012-0095-2 (Cited on page 75).

Zenil, Hector, Jean-Paul Delahaye, and Cédric Gaucherel (2012). « Image Characterization and Classification by Physical Complexity ». In: *Complexity* 17.3, pp. 26–42. ISSN: 1099-0526. DOI: 10.1002/cplx.20388 (Cited on page 75).

Zenil, Hector, Fernando Soler-Toscano, Jean-Paul Delahaye, and Nicolas Gauvrit (Sept. 30, 2015). « Two-Dimensional Kolmogorov Complexity and an Empirical Validation of the Coding Theorem Method by Compressibility ». In: *PeerJ Computer Science* 1, e23. ISSN: 2376-5992. DOI: 10.7717/peerj-cs.23 (Cited on pages 75, 80, 93).

Zenil, Hector and Elena Villarreal-Zapata (Sept. 1, 2013). « Asymptotic Behavior and Ratios of Complexity in Cellular Automata ». In: *International Journal of Bifurcation and Chaos* 23.09, p. 1350159. ISSN: 0218-1274. DOI: 10.1142/S0218127413501599 (Cited on page 75).

RÉSUMÉ

Dans cette thèse, nous explorons l'utilisation de systèmes complexes pour étudier l'apprentissage et l'adaptation dans les systèmes naturels et artificiels. L'objectif est de développer des systèmes autonomes capables d'apprendre sans supervision, de se développer de manière autonome et de devenir de plus en plus complexes avec le temps. Les systèmes complexes apparaissent comme un cadre adapté pour comprendre ces phénomènes en raison de leur capacité à croître en complexité. Être en mesure de construire des algorithmes d'apprentissage qui nécessitent peu ou pas de supervision permettrait une plus grande flexibilité et adaptabilité dans diverses applications. En tentant de comprendre les principes fondamentaux de l'apprentissage dans les systèmes complexes, nous espérons améliorer notre capacité à concevoir et à mettre en œuvre des algorithmes d'apprentissage à l'avenir. Cette thèse apporte les contributions suivantes : le développement d'une métrique de complexité générale que nous appliquons pour rechercher des systèmes complexes qui présentent une croissance de complexité, l'introduction d'une méthode pour étudier les calculs dans des systèmes complexes à grande échelle, et le développement d'une métrique pour l'efficacité de l'apprentissage ainsi que l'introduction d'un jeu de données de référence pour évaluer la vitesse des algorithmes d'apprentissage. Nos résultats contribuent significativement à notre compréhension de l'apprentissage et de l'adaptation dans les systèmes naturels et artificiels. De plus, notre approche contribue à une nouvelle direction prometteuse pour la recherche dans ce domaine. Nous espérons que ces résultats inspireront le développement d'algorithmes d'apprentissage plus efficaces et plus performants à l'avenir.

MOTS CLÉS

Apprentissage automatique ⋆ Intelligence artificielle ⋆ Systèmes complexes ⋆ Complexité

ABSTRACT

In this thesis, we explore the use of complex systems to study learning and adaptation in natural and artificial systems. The goal is to develop autonomous systems that can learn without supervision, develop on their own, and become increasingly complex over time. Complex systems are identified as a suitable framework for understanding these phenomena due to their ability to exhibit growth of complexity. Being able to build learning algorithms that require limited to no supervision would enable greater flexibility and adaptability in various applications. By understanding the fundamental principles of learning in complex systems, we hope to advance our ability to design and implement practical learning algorithms in the future. This thesis makes the following key contributions: the development of a general complexity metric that we apply to search for complex systems that exhibit growth of complexity, the introduction of a coarse-graining method to study computations in large-scale complex systems, and the development of a metric for learning efficiency as well as a benchmark dataset for evaluating the speed of learning algorithms. Our findings add substantially to our understanding of learning and adaptation in natural and artificial systems. Moreover, our approach contributes to a promising new direction for research in this area. We hope these findings will inspire the development of more effective and efficient learning algorithms in the future.

KEYWORDS

Machine learning ⋆ Artificial Intelligence ⋆ Complex systems ⋆ Complexity