Doctoral theses at NTNU, 2023:57

Sidney Pontes-Filho

Optimization of dynamical systems towards criticality and intelligent behavior

Doctoral thesis

NTNU
Norwegian University of Science and Technology
Thesis for the Degree of Philosophiae Doctor
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

NTNU
Norwegian University of
Science and Technology

Sidney Pontes-Filho

Optimization of dynamical systems towards criticality and intelligent behavior

Thesis for the Degree of Philosophiae Doctor

Trondheim, March 2023

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

NTNU
Norwegian University of
Science and Technology

NTNU
Norwegian University of Science and Technology

Thesis for the Degree of Philosophiae Doctor

Faculty of Information Technology and Electrical Engineering
Department of Computer Science

© Sidney Pontes-Filho

ISBN 978-82-326-5746-9 (printed ver.)
ISBN 978-82-326-6953-0 (electronic ver.)
ISSN 1503-8181 (printed ver.)
ISSN 2703-8084 (online ver.)

Doctoral theses at NTNU, 2023:57

Printed by NTNU Grafisk senter

Abstract

With the progress of computing power, artificial intelligence (AI) systems are able to achieve outstanding results that surpass human-level performance on some tasks, such as image recognition. Mainstream AI systems are only able to learn one or more specific tasks, and they commonly fail to go beyond what they were trained for. To have a real breakthrough in the field of AI, those systems require adaptability and the capacity to learn general tasks and data distributions. Such characteristics are among the objectives of artificial general intelligence (AGI) research. The inspiration from biology, neuroscience, and complex systems may guide the development of AGI because AI systems are still rigid instead of self-organizing. Therefore, the research reported in this thesis aims at intelligent dynamical systems that resemble the brain, such as the brain's critical behavior that may allow the cortex to self-organize to criticality in order to increase computational capacity. The plan to accomplish such systems encompasses the usage of optimization methods to find adequate interactions of the components in a complex system, how they are connected with each other, and how they adapt those interactions and connections over time. The dynamical systems investigated in this research are cellular automata, Boolean networks, and recurrent neural networks with abstract neuron models or with more biologically plausible ones (spiking neurons). The main optimization method applied in these systems is evolutionary computation or artificial evolution. There are three parts in the presented research. The first uses a deep neural network library to evolve dynamical systems towards criticality. The second part works on the complexification of spiking neural networks with adaptive synapses for solving tasks in mutable environments. The last one involves the application of neural cellular automata for controlling robots and even developing their morphology with artificial embryogeny. The results of this research attempt to address some unanswered questions related to the practicality, methodology, and benefits of applying dynamical systems in AI.

Preface

This thesis is submitted in partial fulfillment of the requirements for the degree of Philosophiae Doctor (PhD) in Computer Science at the Norwegian University of Science and Technology (NTNU). The work was conducted at and funded by Oslo Metropolitan University (OsloMet), with additional funding from the Norwegian Council under the SOCRATES project, grant number 270961. Professor Stefano Nichele was the main supervisor for the work. Professors Gunnar Tufte, Anis Yazidi, Jianhua Zhang, Hugo Hammer, and Ioanna Sandvig were the co-supervisors.

The thesis is in the form of a collection of papers which have already been published or submitted to workshops, conferences, or journals.

Acknowledgements

I would like to express my deepest gratitude to my supervisor Professor Stefano Nichele for his support, advice, and enthusiasm during this research. I had the extraordinary good fortune of being able to discuss my ambitious ideas with him. He motivated me to pursue them and also helped me a lot in realizing them.

I am also grateful to my co-supervisors Professors Gunnar Tufte, Anis Yazidi, Jianhua Zhang, Hugo Hammer, and Ioanna Sandvig for their valuable guidance and feedback. I would like to thank my co-supervisors in my heart that are Professor Pedro Lind, and Associate Professor Gustavo Mello for the outstanding discussions and work together. I am thankful to my colleagues and friends in the SOCRATES project and the Department of Computer Science at OsloMet for the amazing time during our work, events, trips, and planned social gatherings. I would like to thank Professor Sebastian Risi and everyone in the Robotics, Evolution and Art Lab at the IT University of Copenhagen for the wonderful research visit. I want to place on record my appreciation to the co-authors of all publications mentioned in this thesis. All meetings and discussions were so enjoyable, and I am very proud of the work we have done.

To be motivated during a PhD, one requires the encouragement of caring friends and colleagues. Kristine Heiney joined almost together with me, and I appreciate her support and kindness a lot. I would like to highlight two truly exceptional friends, Akriti Sharma and Marko Stojiljkovic. They are the ones with whom I feel so safe and cared for. I am so thankful to my friends who take part in my routine at work, playing games, and hanging out.

My family and friends in Brazil and around the world have been an enormous support and I am extremely grateful to all of them, especially to my mom and dad, Mércia and Sidney Pontes. Despite the distance, they have been so supportive and loving. It also extends to my brother, Victor, for his friendship, and the good moments. This journey has become so incredible with the ones that have said they love or like me. Finally, my cutest thanks to Rosa Maria, my dog, and my cockatiels for the happiness of being with them. Mere words cannot express my gratitude to my friends who have taken care of my pets when I needed to travel.

Contents

Part I

Research Overview

Chapter 1

Introduction

Artificial intelligence systems have advanced dramatically in the last decades, similarly to the computing power which has pushed forward such advancement. The field of deep neural networks in AI is rising with the support of more powerful and highly parallel hardware such as the graphics processing units (GPUs) and has surpassed human-level performance in image recognition and several control tasks [9]. However, those AI systems are narrow or specialized to solving those tasks. Therefore, a part of the AI community is working on artificial general intelligence, which is inspired by how human intelligence is able to generalize [10, 11]. The current AI systems are rather rigid and do not adapt enough to different situations. A proposed alternative for improving AI is by using complex dynamical systems, which can be adaptive and even contain self-organizing and self-assembling properties. Successfully exploiting complex dynamical systems in AI may pave the way towards intelligent behavior that is more akin to what is found in nature [12].

It is both a dream and a challenge to have artificial systems similar to biological ones, particularly for intelligence. In order to do that, the research community interested in biologically inspired artificial intelligence needs to deal with interdisciplinarity, as there are many disciplines involved in this quest. The main ones are AI and neuroscience. However, there are other key disciplines, such as complex systems, evolutionary computation, and artificial life. In essence, the study of AI is undertaken to achieve precise descriptions of learning or intelligence, so a machine can be created to simulate those descriptions [13, 14]. Neuroscience studies the structure and function of the brain [15]. A complex system consists of a network of components interacting in a nonlinear manner and has two main properties: self-organization and emergence [16]. Evolutionary computation is a population-based search method applied for optimization and inspired by natural evolution [17]. The goal of artificial life is to synthesize and simulate biological systems [18].

This thesis is focused on the optimization of artificial dynamical systems for achieving behaviors similar to those found in the brain, such as a dynamical regime called criticality that is hypothesized to arise via self-organization and enhance computational capacity in the cortex [19]; and also intelligence itself. This thesis therefore is interdisciplinary and covers all the aforementioned disciplines. The dynamical systems used in this thesis include cellular automata [20], Boolean networks [21], and recurrent neural networks with perceptron [22, 23] or spiking neuron models [24, 25]. Dynamical systems are mainly optimized with evolutionary computation methods, such as genetic algorithms [26], evolution strategy [27], and MAP-elites [28], and also a reinforcement learning algorithm, which is the deep Q-learning [29].

The research in this work is divided into three parts. The first is the development of a framework that leverages the power of GPUs to simulate evolvable dynamical systems towards criticality and evaluate their robustness against noise. The second part is the usage of genetic algorithms to grow weight agnostic spiking neural networks with neuroplasticity. In this way, the evolved networks may present adaptability and generalization in AI tasks. The final part is the control of robots with a cellular automaton variant, which is updated by an artificial neural network and may present self-organization depending on the trained task. The name of such systems is neural cellular automata (NCAs) [30–32].

1.1 Research questions

Artificial intelligence systems based on artificial neural networks are often said to be inspired by their biological counterparts. However, this inspiration is rather abstract [14]. The details of the inner workings of biological neurons and their surroundings are often neglected, together with the dynamical nature of neuronal populations and the way they learn. Intrinsically, deep learning relies on rigid systems that are in fact quite far removed from their biological inspiration [12]. Neuroscience has established that the brain is better described as a dynamical system with a dynamical state that is self-organizing through local interactions of the nerve cells. In particular, there is evidence that a state that allows for higher computational capacity is often reached, named criticality [19, 33]. In AI, such properties of self-organization and criticality are rarely considered and therefore there is a knowledge gap on how to incorporate such properties in artificial neural networks. As such, the novel goal of this research was to explore how the concepts of self-organization and criticality may be applied to AI systems. The work in this thesis is explorative and experimental and aims at establishing the foundation of criticality and self-organization in AI with the motivation of inspiring further research on the topic. With that in mind, the following research questions have been formulated:

Research question 1

How can dynamical systems be optimized towards criticality?

Criticality is a dynamical state in complex systems. In this state, the system may have two modes. One is to be in the vicinity of a phase transition between order and chaos. This mode is tunable by one or more parameters and allows for the maintenance of complex patterns of activity over a wide range in space and time. This is also known as the "edge of chaos", and it is said to support computational capacity through processing, transmission and storage of information [19, 34]. The other mode is an infinite correlation between space and time where information similarly spreads through several scales. Thus, criticality presents a power law distribution over several orders of magnitude, resembling a fractal structure that is similar over different scales. Different from the first mode, the system may tune itself towards the critical state independent of its initialization. This is called self-organized criticality and it is hypothesized that it increases computational capacity in the cortex [3, 19, 33, 35].

Dynamical systems can have subcritical, critical, or supercritical behavior, depending on how synchronous the components of the system are, ranging from loosely coupled systems with random activity (subcritical) to strongly coupled systems with highly synchronous activity (supercritical). A system in criticality is in a small region of the dynamical state space where there is an average synchronization [19, 36]. This small region for criticality occurs when the complex system has proper network topology, communication rules, and plasticity. The plan to address this research question is to optimize at least one of these three characteristics of a dynamical system using a fitness function that measures how critical its dynamics is.

Research question 2

How can dynamical systems be optimized to present adaptable and intelligent behavior?

The most common methods in AI are not dynamical systems. As such, they often lack adaptation and are not self-organizing. The meaning of adaptable can be understood as that characteristic of a component of a system whereby it changes how it communicates depending on its previous states. In other words, it is plastic, and the same input of a component can produce a different outcome over time. This is also a feature that biological neurons have for adaptation and learning, which is known as neuroplasticity. In a biological neural network, such a feature modifies the synaptic strength of the connections and also creates and prunes them, but it is not limited to that [37].

Intelligence is still hard to define [38, 39]. Nevertheless, Minsky [40] states that intelligent behavior is typically the capability to solve hard problems. For addressing this research question, intelligence is defined as the ability to act accordingly depending on the state of the environment. The tasks to measure the intelligence of the optimized dynamical systems are the ones that require supervised learning and reinforcement learning in AI [41]. Related to adaptability, the tasks need to be designed to favor adaptable systems; for example, performing a task in a mutable environment. The main system to be optimized is an artificial neural network with perceptron or spiking neuron models. The optimization can affect the network structure, the weights of the connections, and the neuroplasticity parameters.

Research question 3

Are there benefits of using self-organizing dynamical systems in artificial intelligence? If so, what are they?

Self-organization is a dynamical process in complex systems that governs them towards nontrivial macroscopic structures and/or behaviors over time [16]. This encompasses the co-evolution of both structure and function, where the system is its own designer. This functional organization may provide the system with stability, adaptability, and autonomy, all properties desirable in AI systems. Moreover, such properties are considered to be at the foundation of AGI. This research question can be addressed by analyzing and testing the features of artificial intelligence systems that can show self-organization.

1.2 Associated project

The research in this thesis is associated with the project named SOCRATES[1] (Self-Organizing Computational substRATES). It is funded by the Norwegian Research Council (NFR), grant number 270961. SOCRATES is a long-term project seeking the creation of cutting-edge data analysis with dynamical hardware. It aims to create a theoretical and experimental foundation for a new computing paradigm by exploiting novel, self-organizing, robust, efficient, and unconventional dynamical substrates. SOCRATES works with two physical substrates, biological and artificial. The biological substrates are in vitro biological neural networks interconnected to microelectrode arrays (MEAs) and the artificial ones are ensembles of nanomagnets.

1.3 Thesis outline

This thesis is a compilation of papers and is divided into two parts. The first and current part is the "Research Overview". It is organized as follows: Chapter 2 consists of the theoretical background necessary to understand the performed research. Chapter 3 explains the research process related to the published work. It also describes the ideas and relationships between each work. Chapter 4 presents the summary of the research results in the form of an abstract and analysis of the published work. Chapter 5 is a discussion of the

[1]`https://www.ntnu.edu/socrates`

research questions in relation to the research results. Chapter 6 concludes the thesis.

The second part of the thesis is the "Publications", which includes all the papers summarized in Chapter 4.

Chapter 2

Background

This thesis covers several disciplines, and this chapter explains the background theory needed to understand the concepts and approaches applied to perform the research described in the next chapters, and in the publications.

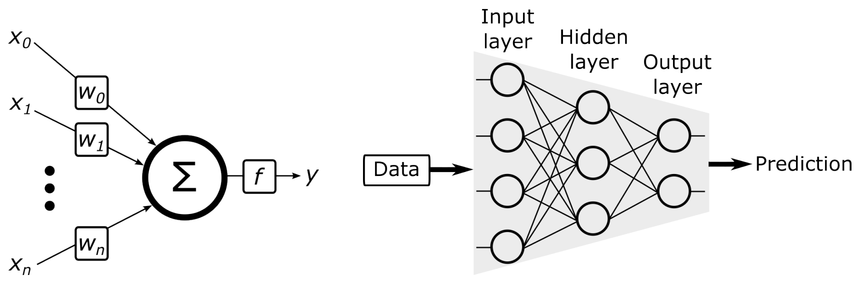
2.1 Artificial intelligence

The research field of AI may be considered a branch of computer science and covers a broad range of methods. This field studies the simulation of intelligent behavior [42]. This section describes the part of AI that is relevant to this thesis.

2.1.1 Artificial neural networks

The perceptron is the first neuron model. Its theory was introduced in 1943. Afterwards, many theorists have discussed brain models. The origin of the artificial neural network came from those discussions and one of the purposes of the artificial neural network was to resemble the synapses in the brain [14, 43–46].

The perceptron is a function where the input vector x with n elements is multiplied in an element-wise manner with the weight vector w, also with n elements. After this multiplication, the resulting vector is aggregated with summation, which gives a scalar that is then applied to an activation function f.



(a) Perceptron **(b)** Multilayer perceptron

Figure 2.1: Illustration of a perceptron and a multilayer perceptron. (a) The equation of this perceptron is described by (2.1). (b) This multilayer perceptron has only one hidden layer, but there can be more.

The output of the perceptron is y and the mathematical formula of this process is presented by

$$y = f(\sum_{i=0}^{n} x_i w_i). \tag{2.1}$$

The activation function f was initially theorized as a binary threshold or step function, such as

$$f(s) = \begin{cases} 1 & s > \beta \\ 0 & s \leq \beta, \end{cases} \tag{2.2}$$

where β adjusts this threshold and can be understood as the bias of the perceptron. The bias can also be a constant input of value $+1$, and its adjustment is performed by changing its connection weight. After introduction of the perceptron, other activation functions were described, such as sigmoid, hyperbolic tangent, and rectified linear unit.

One example of an artificial neural network is the multilayer perceptron. The architecture of the multilayer perceptron consists generally of an input layer for stimulating the next layer with the data sample, a defined number of hidden layers, and an output layer of perceptrons. The communication happens sequentially through the layers from input to output [47]. Fig. 2.1 depicts a perceptron and a multilayer perceptron. When a multilayer perceptron has a depth of 5 to 20 layers, the network can learn multiple levels of data representation and also how to extract features from raw data. This method presents outstanding results in applications ranging from image recognition to natural language processing. Such a multilayer perceptron is the basis of the promising research field of deep learning [41].

2.1.2 Spiking neural networks

A more biologically plausible artificial neural network is one consisting of spiking neurons. This neuron model resembles biological neurons, and their communication in continuous time with action potentials or spikes through synapses [48, 49]. The data type processed in a spiking neural network is a time series of binary values, which indicates whether there is a spike or not. The spiking neuron has a state named membrane potential that integrates the synaptic inputs. Once the membrane potential reaches a value above a threshold, this neuron produces a spike. The spikes can be excitatory or inhibitory, accordingly increasing or decreasing the membrane potential. Consequently, it alters the likelihood of the neuron generating an action potential. Depending on the spiking neuron model, the membrane potential can have several dynamics. A commonly applied model of spiking neuron is the leaky integrate-and-fire. This is a very efficient model, and the membrane potential of the cell has a leakage that exponentially decreases the membrane potential towards its resting value.

2.1.3 Reservoir computing

In the field of machine learning, a reservoir is a dynamical system with a certain behavior. This system may expand the dimensionality of the input data. The combination of proper dynamics and dimensionality expansion allows for a linear machine learning technique to (partially) read the state of the reservoir and then solve tasks with nonlinear input data [50]. When reservoir computing was introduced, the initial idea was to use a randomly connected recurrent neural network to be the reservoir. Such an introduction was made by two different research groups independently and almost simultaneously. Jaeger [22] introduced the echo state network, which is a recurrent neural network with perceptron, while Maass et al. [24] proposed the liquid state machine that consists of spiking neurons. Fig. 2.2 shows a reservoir computing example with

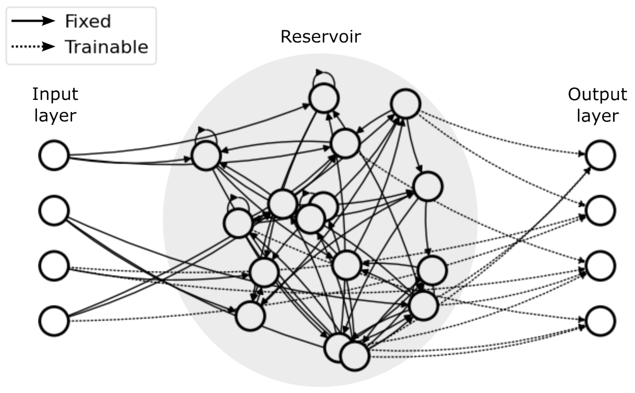


Figure 2.2: Illustration of reservoir computing with a randomly connected recurrent neural network. The trainable connections are only the ones that go to the linear readout layer or output layer. In this example, the readout layer also reads some neurons in the input layer.

a randomly connected recurrent neural network. The linear machine learning method is a single neural layer indicated by the output layer. Therefore, the connections to the output layer are the ones that are trainable.

After introduction of these two reservoir computing methods more types of reservoirs were described, even with physical substrates. Such physical dynamical systems can be an analog circuit, a nanomagnet ensemble, a neuronal culture over microelectrode arrays, and surprisingly even a bucket of water [51, 52].

2.1.4 Artificial general intelligence

Most of the research in AI is to develop very specialized methods, aiming at ever better performance on the specific task. On the other hand, a part of the research community works on general-purpose systems instead of special-purpose ones. This area of research is known as artificial general intelligence, also called strong AI. One possible idea of AGI could be the combination of many specialized methods. However, most of the AGI community believes that its difference from conventional AI is qualitative rather than quantitative

[39, 53]. With the advent of AGI many technologies will be possible, such as self-driving cars, conversational assistants, and housekeeping robots [11].

2.2 Dynamical systems

Dynamical systems are defined as systems that change their states over time following predefined rules. The traditional definition of dynamical systems states that the rules are deterministic. However, stochastic systems can also be modeled [16].

There are some dynamical systems that are complex. Such systems contain simple components that interact among themselves nonlinearly. Therefore, they are capable of producing novel properties in the macroscopic collective behavior and spontaneously form distinctive structures in time, space, and function [54].

Examples of complex systems are very noticeable in everyday life. The human body is a collection of complex dynamical systems, such as the nervous system and immune system. Our society and ecosystems are also other examples, but on a larger scale. These systems consist of many interacting elements that form a network whose behavior goes beyond that of its constituent parts [54].

2.2.1 Cellular automata

A popular type of complex system is the cellular automaton (CA). This system is formed by cells spatially distributed in a regular grid. Such a grid can have any number of spatial dimensions, but the most researched CAs have one or two dimensions. The cells have states that are normally binary but can even be ternary or a larger and finite n-ary. CAs with continuous states exist as well. The states of all cells in the CA change uniformly and synchronously over discrete time. These changes are coordinated by a state-transition function, which considers the local information given by the states of a cell and its neighbors. The application of the state-transition function can also be asynchronous and there are heterogeneous CAs whose cells can be updated with different state-transition functions [16, 20, 55].

CA was discovered by John von Neumann and Stanislaw Ulam in the 1940s. Von Neumann was working on an artificial life project aimed at creating a self-reproducing automaton. While he had an issue with his initial model, Ulam gave him some advice because of his work on crystal growth. Subsequently, they together developed a two-dimensional abstract universe with locality properties, and discrete time and space [16, 56, 57].

2.2.2 Random Boolean networks

A random Boolean network (RBN) consists of a cellular automaton with an irregular grid. Therefore, the nodes in an RBN are randomly connected. This configuration makes RBNs a generalization of cellular automata. Therefore, it broadens the research performed in artificial life. Stuart Kauffmann created RBNs for modeling genetic regulatory networks. The advantage of this dynamical system is its generic properties that allow for the modeling of systems with complex and unknown connectivity [21].

2.2.3 Randomly connected recurrent neural networks

Recurrent connections in an artificial neural network add a dimension of time to the system. Therefore, recurrent neural networks seem ideal to solve machine learning tasks with temporal or sequential data. Buonomano and Merzenich [58] introduced a randomly connected recurrent neural network with spiking neurons and short-term plasticity. This network is kept untrained while a readout layer is trained with a correlation-based learning rule. This method was the basis for the introduction of reservoir computing (described in Section 2.1.3), as the echo state networks and liquid state machines [50, 59].

2.3 Criticality

Complex dynamical systems may be in a dynamical state called "criticality", a state that may support computation, depending on some favorable conditions. Such conditions may be found by tuning the system parameters or setting adequate configurations, such as the state-transition function, topology, and plasticity. Langton [34] introduced a tuning parameter for 1D cellular automata. This parameter is interpreted as the "temperature" of the system and is named as λ parameter. This tunable parameter also measures the computability of the substrate. In other words, it quantifies the capacity of the system to transmit, store, and modify information.

In a synchronous and homogeneous 1D CA with the number of possible cell states as S, the number of neighbors as N, and the number of transitions to a chosen state as n while the other transitions in the state-transition rule are defined randomly with the remaining states, the λ parameter is calculated with

$$\lambda = 1 - \frac{n}{S^N}. \tag{2.3}$$

The chosen state counted by n is referred to as the quiescent state and n/S^N indicates the ratio of transitions to this state. If $\lambda = 0$, all transitions result in the quiescent state. Thus, all cells in the CA change to this state in the first application of the state-transition function. If $\lambda = 1$, it means that $n = 0$ and the transitions are randomly selected to change to the remaining (non-quiescent) states. To equally represent all states in the state-transition function, it results in $\lambda = 1 - 1/K$. Therefore, $\lambda = 0$ is the most ordered state-transition function and $\lambda = 1 - 1/K$ is the most chaotic one. Wolfram [60] introduces a classification system for the behavior of the CAs. This system consists of four classes, and they are:

- *Class I* means that the CA is static.
- *Class II* presents cyclic or periodic global states over time.
- *Class III* has chaotic behavior.
- *Class IV* presents complex structural patterns, which may support computation.

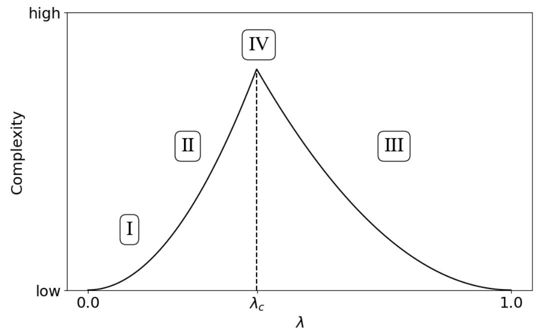


Figure 2.3: Location of λ_c and the four behavior classes of Wolfram in the λ space over an abstract complexity measurement (Adapted from Ref. [34]).

Order and chaos are the two phases in the CA. Order is represented by classes I and II, and chaos by class III. If a CA presents these phases at the same time as complex structures emerge, then the system is in the vicinity of a phase transition or at the "edge of chaos". Therefore, the system behaves as class III, so that information can travel long distances without decaying rapidly into random noise, which supports computability.

The vicinity of a phase transition is characterized by a critical point of a control parameter. Such a point can be located in the space of the λ parameter and is represented by λ_c. Fig. 2.3 exemplifies the location of λ_c and Wolfram's classes in the λ space over an abstract complexity measurement.

Another type of criticality is analyzed by Bak et al. [35]. In this analysis, the behavior of a critical system presents in several scales a power law distribution $p(x) \propto x^{-\alpha}$ of noise (in time) or structure (in space), or even in both [61]. Since it is a power law distribution, the behavior of the system is similar over different scales, such as a fractal.

If the critical point is an attractor, it is considered that the system self-organizes towards criticality. This self-organization happens independently of the current state of the dynamical system, even when it is far from stable states. This concept is known as self-organized criticality and can be illustrated with a sandpile metaphor [62]. When one slowly adds sand grains to the pile, it will trigger avalanches of any size and duration with the addition of even one grain. The distributions of avalanche size and duration follow a power law and span many orders of magnitude depending on the system size. The sandpile has a critical angle of the slope that causes avalanches. Therefore, the pile self-organizes to maintain a constant angle.

2.4 Artificial development

The development of multicellular organisms represents the capacity of a system to compress its morphology, functionality, and maintenance in a genetic code. In the research field of artificial life, this capacity is seen as advantageous for artificial systems, particularly for scalability because the complexity and information of an organism's phenotype are much greater than its genome. Such a difference is generally described as genomic bottleneck [63, 64]. Therefore, a subfield in artificial life is introduced for replicating those biological features and it is known as artificial development, and also called artificial embryogeny and artificial ontogeny [65–67].

Some methods in artificial development apply cellular automata for their models. This approach simulates the cell chemistry during biological development [65]. A recently popular type of cellular automaton in artificial development is the neural cellular automaton, which uses an artificial neural network to define the state-transition function. In 2017, an NCA was used to develop 2D patterns and trained with artificial evolution [31]. More recent NCAs are differentiable and then optimized with gradient descent. The NCA introduced by Mordvintsev et al. [32] develops a target image from a single active cell in the grid. The update rule is asynchronous and the NCA can regenerate the developed image when parts are erased. Another differentiable NCA grows 3D structures from a "seed" cell [68], which also presents regeneration properties. This method goes beyond 3D artifacts and can develop functional machines. That is possible because some components of the structure are dynamic.

The common approach to implementing an NCA is illustrated in Fig. 2.4. The artificial neural network in the NCA receives as an input the neighborhood of the cell to be updated. The output of the network is a value that is summed together with the previous state of the grid after all cells are processed. However, the update may only occur in the cells depending on a mask. This mask

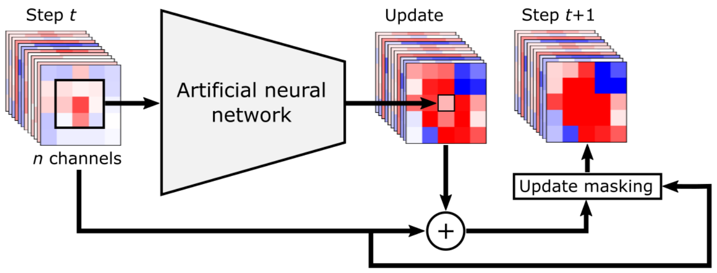


Figure 2.4: Illustration of neural cellular automata. All cells in the grid are updated and their neighborhoods are adjusted to center the updated cell. Update masking defines the cells that are going to be updated in the next time step (Adapted from Paper C.2).

is related to the cells that can update due to their state, which may be considered alive or able to grow, or to a probability that emulates the asynchronicity in biological systems and also makes the NCA more robust.

2.5 Artificial evolution

Biological life has evolved from single cells to animals with intelligent behavior. In nature, evolution has generated numerous species since the first living being. Due to natural selection and other factors, some are thriving, and others are already extinct [69]. Therefore, the optimization that is observed in evolution is a population-based method of trial-and-error [70].

In computer science, artificial evolution is a simulation of this natural process, and so it can be used as an optimization method or a partial search algorithm that would find solutions to defined tasks and problems. The general scheme of artificial evolution or evolutionary computing starts with a population of random solutions, which are tested through a benchmark method to measure a fitness score; part of the population is selected based on the fitness score, and these selected individuals reproduce with some variations a new generation of solutions. This cycle of testing and reproducing solutions continues until

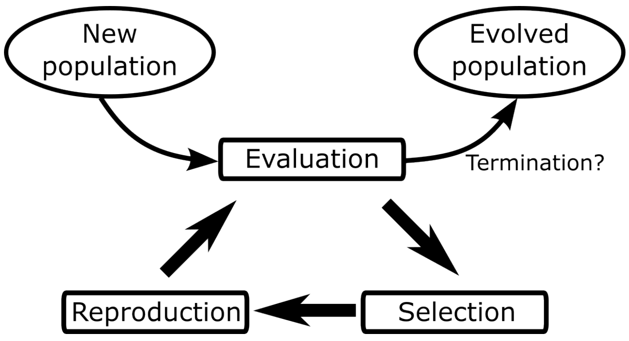


Figure 2.5: General scheme of artificial evolution. Reproduction occurs with a variation of the selected individuals.

a terminal condition is met, or a maximum possible number of generations is reached. Fig. 2.5 illustrates this general process.

The different methods in artificial evolution differ in what their genomes encode, such as data in general, real numbers in evolution strategy [27], or computer code in genetic programming [71]. There are varied approaches for parent selection and reproduction. In genetic algorithm [26], the reproduction may involve the recombination of the genetic code (or parameters) of the parents, and also mutation of their genes. In evolution strategy, the new generation is sampled from a multivariate normal distribution calculated with the best individuals in the current generation. In MAP-elites [28], the parents are selected from a population of the best solutions in niches based on specified features.

Chapter 3

Research process

This chapter explains the research process during the four-year PhD program at the Department of Computer Science, Faculty of Information Technology and Electrical Engineering, NTNU. The research was mostly conducted at OsloMet because it was initially the primary institution of my main supervisor as well as a research partner in the SOCRATES project. The four-year PhD program consists of full-time research work related to the PhD, with 25% of teaching and administration duties. This thesis only includes the research work performed during the PhD.

The 12 scientific publications produced in these four years are divided into four categories. Table 3.1 summarizes these categories, presenting their topics and number of papers. Fig. 3.1 illustrates a timeline with the logical connections of the publications.

Category	Topic	#papers
A	EvoDynamic: general representation of dynamical systems and their evolution towards criticality	3
B	NeuroEvolution of low-level artificial general intelligence	2
C	Neural cellular automata for control tasks	3
D	Papers not included in the thesis	4

Table 3.1: Paper categories

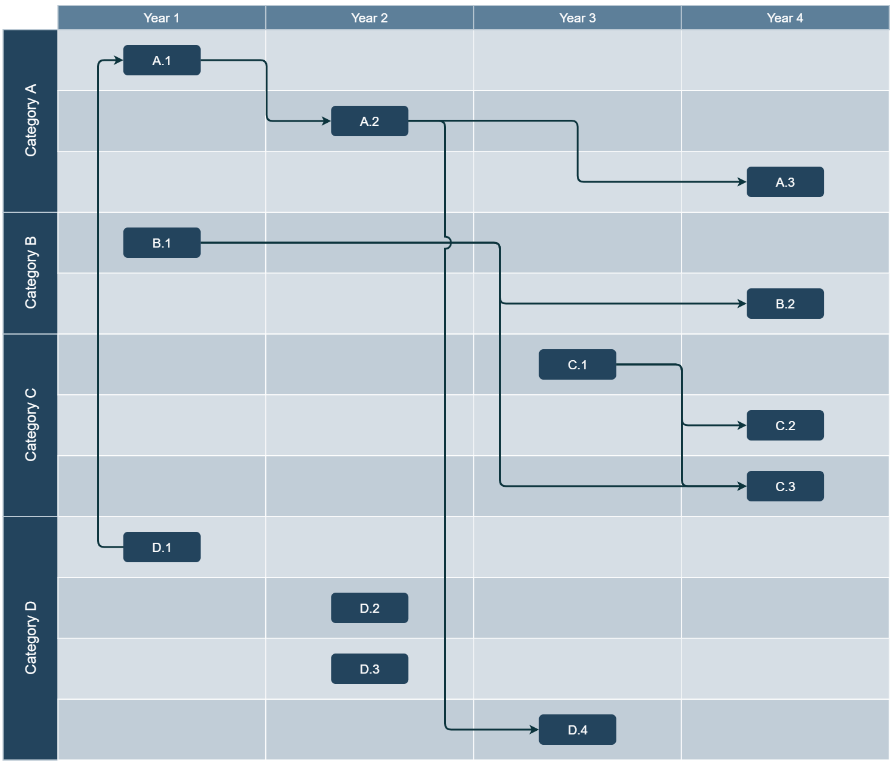


Figure 3.1: Timeline of the publications with their logical connections.

3.1 Paper category A

The paper category A is related to the development and application of the Python library, EvoDynamic. This library has a general representation of dynamical systems based on artificial neural networks. Therefore, it implements dynamical systems in a deep learning framework, such as TensorFlow, to allow for GPU usage. An application of EvoDynamic was to evolve towards criticality cellular automata, random Boolean networks, and echo state networks (Papers A.1 and A.2). Those dynamical systems could be deterministic or stochastic. EvoDynamic was also applied to verify the robustness of criticality in a critical stochastic cellular automaton (Paper A.3). The main objective of this category is to answer research question 1. The papers in this category are presented in Table 3.2.

ID	Title	Ref.
A.1	EvoDynamic: a framework for the evolution of generally represented dynamical systems and its application to criticality	[1]
A.2	A neuro-inspired general framework for the evolution of stochastic dynamical systems: cellular automata, random Boolean networks and echo state networks towards criticality	[2]
A.3	Assessing the robustness of critical behavior in stochastic cellular automata	[3]

Table 3.2: Paper category A

3.2 Paper category B

The paper category B is fully related to the method called neuroevolution of artificial general intelligence (NAGI). Paper B.1 describes the concept of NAGI. After that, NAGI is implemented, and its results are presented in Paper B.2. One of the objectives of this category is to address research questions 2 and 3. Table 3.3 presents the papers in this category.

ID	Title	Ref.
B.1	A conceptual bio-inspired framework for the evolution of artificial general intelligence	[4]
B.2	Towards the neuroevolution of low-level artificial general intelligence	[5]

Table 3.3: Paper category B

3.3 Paper category C

The paper category C corresponds to the usage of neural cellular automata for control tasks. This dynamical system is known for its capability of presenting self-organization when trained in certain tasks where this feature is useful or required. Paper C.1 applies NCA to control a cart-pole agent. Paper C.2 describes a single NCA that can develop a robot body from a single cell through morphogenesis [72]. Paper C.3 describes a spiking NCA for the distributed control of voxel-based soft robots. The research performed in this category is also meant to address research questions 2 and 3.

ID	Title	Ref.
C.1	Towards self-organized control: Using neural cellular automata to robustly control a cart-pole agent	[6]
C.2	A unified substrate for body-brain co-evolution	[7]
C.3	Collective control of modular soft robots via embodied spiking neural cellular automata	[8]

Table 3.4: Paper category C

3.4 Paper category D

The paper category D is the group of papers not included in this PhD thesis. Of these, Ref. [73] is the preliminary work for Category A. It was published at a workshop. Ref. [74] is an extended abstract that gives an update of the work performed in the SOCRATES project that is mainly related to Category A. It was also published at a workshop and describes the EvoDynamic framework (from Paper A.2), the work with in vitro biological neuronal networks on microelectrode arrays, and the development of computational hardware made of two-dimensional arrays of coupled nanomagnets, which is called artificial spin ice. Refs. [75, 76] were published in conferences and are related to the application of deep learning methods for neuroscience. Table 3.5 shows a summary of this paper category.

ID	Title	Ref.
D.1	A general representation of dynamical systems for reservoir computing	[73]
D.2	Method to obtain neuromorphic reservoir networks from images of in vitro cortical networks	[75]
D.3	A deep learning-based tool for automatic brain extraction from functional magnetic resonance images of rodents	[76]
D.4	Bridging the computational gap: From biological to artificial substrates	[74]

Table 3.5: Paper category D

Chapter 4

Research result

This chapter is an overview of the papers resulting from the research, which have been included in this thesis. Each section consists of the title of the work, the authors, and the year and place of the publication. These are followed by an abstract of the paper, the contributions of every author, and also the story behind the publication with its reflections.

4.1 Paper A.1

EvoDynamic: a framework for the evolution of generally represented dynamical systems and its application to criticality

By *Sidney Pontes-Filho, Pedro Lind, Anis Yazidi, Jianhua Zhang, Hugo Hammer, Gustavo B. M. Mello, Ioanna Sandvig, Gunnar Tufte and Stefano Nichele (2020)*

Published in *EvoApplications 2020: International Conference on the Applications of Evolutionary Computation (Part of EvoStar)*

Dynamical systems possess a computational capacity that may be exploited in a reservoir computing paradigm. This paper presents a general representation of dynamical systems which is based on matrix multiplication. That is similar to how an artificial neural network (ANN) is represented in a deep learning library and its computation can be faster because of the optimized matrix operations that such a type of library has. Initially, we implement the simplest dynamical system, a cellular automaton. The mathematical fundamentals behind an ANN are maintained, but the weights of the connections and the activation function are adjusted to work as an update rule in the context of cellular automata. The advantages of such implementation are its usage on specialized and optimized deep learning libraries, the capabilities to generalize it to other types of networks and the possibility to evolve cellular automata and other dynamical systems in terms of connectivity, update and learning rules. Our implementation of cellular automata constitutes an initial step towards a more general framework for dynamical systems. Our objective is to evolve such systems to optimize their usage in reservoir computing and to model physical computing substrates. Furthermore, we present promising preliminary results toward the evolution of complex behavior and criticality using genetic algorithm in stochastic elementary cellular automata.

Role of the authors:

Pontes-Filho had the main idea, started the development of the framework with an application, and wrote the paper. Nichele mainly supervised and reviewed the work together with Lind, Yazidi, Zhang, Hammer, Mello, Sandvig and Tufte.

Story and reflections:

This conference paper is an extension of a previous paper published at a workshop (Ref. [73]). It is also the beginning of the development of the Python library called EvoDynamic. The library had only the implementation of cellular automata, which could be 1D or 2D, and deterministic or stochastic; and genetic algorithms for their optimization, which were applied towards criticality. EvoDynamic makes it possible to have any dimensionality and neighborhood in a CA, but it requires a new implementation of a procedural method to create the adjacency matrix of the desired dynamical system. The fitness function optimized with genetic algorithm for criticality was successful, but some adjustments were performed in the function for the next work.

4.2 Paper A.2

A neuro-inspired general framework for the evolution of stochastic dynamical systems: cellular automata, random Boolean networks and echo state networks towards criticality

By *Sidney Pontes-Filho, Pedro Lind, Anis Yazidi, Jianhua Zhang, Hugo Hammer, Gustavo B. M. Mello, Ioanna Sandvig, Gunnar Tufte and Stefano Nichele (2020)*

Published in *Cognitive Neurodynamics*

Although deep learning has recently increased in popularity, it suffers from various problems including high computational complexity, energy greedy computation, and lack of scalability, to mention a few. In this paper, we investigate an alternative brain-inspired method for data analysis that circumvents the deep learning drawbacks by taking the actual dynamical behavior of biological neural networks into account. For this purpose, we develop a general framework for dynamical systems that can evolve and model a variety of substrates that possess computational capacity. Therefore, dynamical systems can be exploited in the reservoir computing paradigm, i.e., an untrained recurrent nonlinear network with a trained linear readout layer. Moreover, our general framework, called EvoDynamic, is based on an optimized deep neural network library. Hence, generalization and performance can be balanced. The EvoDynamic framework contains three kinds of dynamical systems already implemented, namely cellular automata, random Boolean networks, and echo state networks. The evolution of such systems towards a dynamical behavior, called criticality, is investigated because systems with such behavior may be better suited to do useful computation. The implemented dynamical systems are stochastic and their evolution with genetic algorithm mutates their update rules or network initialization. The obtained results are promising and demonstrate that criticality is achieved. In addition to the presented results, our framework can also be utilized to evolve the dynamical systems connectivity, update and learning rules to improve the quality of the reservoir used for solving computational tasks and physical substrate modeling.

Role of the authors:

Pontes-Filho had the main idea, implemented the project, conducted the experiments, and wrote the manuscript. Nichele, Lind, Yazidi, Zhang, Hammer, Mello, Sandvig, and Tufte contributed with supervision, discussions, and reviewing the manuscript.

Story and reflections:

This work is an extension of the previous paper (Ref. [1]), which was an extension of a workshop paper that is not included in this thesis (Ref. [73]). This workshop paper received an invitation into be extended to a journal manuscript for Cognitive Neurodynamics and was accepted to be published. Ref. [73] describes the general framework of EvoDynamic, Paper A.1 includes the application of EvoDynamic to evolve stochastic cellular automata towards criticality, and Paper A.2 adds the evolution of stochastic random Boolean networks and echo state networks towards criticality. In this paper, the fitness function for guiding the evolution to find critical systems was modified. This modification adds the log-likelihood ratio that makes a comparison between the power-law model and the exponential model related to their estimation of the empirical probability distribution of the avalanches in the system. Calculating this ratio is computationally expensive and it is performed only when the temporary fitness score reaches a certain threshold. Even though the fitness functions for criticality of Papers A.1 and A.2 are slightly different, the resulting cellular automata of both works are very similar.

4.3 Paper A.3

Assessing the robustness of critical behavior in stochastic cellular automata

By *Sidney Pontes-Filho, Pedro Lind and Stefano Nichele (2022)*

Published in *Physica D: Nonlinear Phenomena*

There is evidence that biological systems, such as the brain, work at a critical regime robust to noise, and are therefore able to remain in it under perturbations. In this work, we address the question of robustness of critical systems to noise. In particular, we investigate the robustness of stochastic cellular automata (CAs) at criticality. A stochastic CA is one of the simplest stochastic models showing criticality. The transition state of stochastic CA is defined through a set of probabilities. We systematically perturb the probabilities of an optimal stochastic CA known to produce critical behavior, and we report that such a CA is able to remain in a critical regime up to a certain degree of noise. We present the results using error metrics of the resulting power-law fitting, such as Kolmogorov-Smirnov statistic and Kullback-Leibler divergence. We discuss the implication of our results in regards to future realization of brain-inspired artificial intelligence systems.

Role of the authors:

Pontes-Filho and Lind discussed the initial idea and wrote the paper. Pontes-Filho implemented and executed all experiments. Nichele contributed with discussions, ideas, and corrections to the paper.

Story and reflections:

With the evolved stochastic cellular automata from Paper A.2, the probabilities found for this critical system were perturbed with an adjusted Gaussian noise. This adjustment keeps the perturbed probabilities inside the valid range between 0 and 1. The avalanche definition was modified for this paper, but it still keeps the power-law distribution of the avalanches. This new definition makes the avalanches in cellular automata closer to the neuronal avalanches defined in the field of neuroscience.

4.4 Paper B.1

A conceptual bio-inspired framework for the evolution of artificial general intelligence

By *Sidney Pontes-Filho and Stefano Nichele (2019)*

Published in *The 3rd Special Session on Biologically Inspired Parallel and Distributed Computing, Algorithms and Solutions (BICAS 2020) at The 18th International Conference on High Performance Computing and Simulation (HPCS 2020)*

In this work, a conceptual bio-inspired parallel and distributed learning framework for the emergence of general intelligence is proposed, where agents evolve through environmental rewards and learn throughout their lifetime without supervision, i.e., self-learning through embodiment. The chosen control mechanism for agents is a biologically plausible neuron model based on spiking neural networks. Network topologies become more complex through evolution, i.e., the topology is not fixed, while the synaptic weights of the networks cannot be inherited, i.e., newborn brains are not trained and have no innate knowledge of the environment. What is subject to the evolutionary process is the network topology, the type of neurons, and the type of learning. This process ensures that controllers that are passed through the generations have the intrinsic ability to learn and adapt during their lifetime in mutable environments. We envision that the described approach may lead to the emergence of the simplest form of artificial general intelligence.

Role of the authors:

Pontes-Filho had the main idea and mostly wrote the paper together with Nichele. Nichele contributed with discussions, ideas, and corrections to the paper.

Story and reflections:

The main idea of the conceptual method described in this paper came when I properly learned for the first time about artificial neural networks with perceptron and how they learn. This happened in the course "Introduction to Artificial Intelligence" in 2011 during my bachelor’s. I was taught that they are trained through a statistical method called backpropagation of errors, which adjusts their weights and biases according to the mistakes that the network makes. I was not satisfied with this solution, and I wondered if there must be a more realistic way to imitate how our brains memorize and learn new skills. My idea was to use more biologically inspired neuron models with neuroplasticity and optimize those neural networks through artificial evolution. Moreover, the tasks should be simple and require a small adaptable brain to solve them. This can provide a basis for making more complex artificial brains for more complex tasks.

4.5 Paper B.2

Towards the neuroevolution of low-level artificial general intelligence

By *Sidney Pontes-Filho, Kristoffer Olsen, Anis Yazidi, Michael A. Riegler, Pål Halvorsen and Stefano Nichele (2022)*

Published in *Frontiers in Robotics and AI*

In this work, we argue that the search for Artificial General Intelligence (AGI) should start from a much lower level than human-level intelligence. The circumstances of intelligent behavior in nature resulted from an organism interacting with its surrounding environment, which could change over time and exert pressure on the organism to allow for learning of new behaviors or environment models. Our hypothesis is that learning occurs through interpreting sensory feedback when an agent acts in an environment. For that to happen, a body and a reactive environment are needed. We evaluate a method to evolve a biologically-inspired artificial neural network that learns from environment reactions named Neuroevolution of Artificial General Intelligence (NAGI), a framework for low-level AGI. This method allows the evolutionary complexification of a randomly-initialized spiking neural network with adaptive synapses, which controls agents instantiated in mutable environments. Such a configuration allows us to benchmark the adaptivity and generality of the controllers. The chosen tasks in the mutable environments are food foraging, emulation of logic gates, and cart-pole balancing. The three tasks are successfully solved with rather small network topologies and therefore it opens up the possibility of experimenting with more complex tasks and scenarios where curriculum learning is beneficial.

Role of the authors:

Pontes-Filho had the main idea, supervised most of the work, implemented additional experiments, and wrote most of the paper. Olsen contributed to the writing of the paper, performed most of the experimental work as part of his master's thesis while being mainly supervised by Pontes-Filho. Yazidi, Riegler, Halvorsen, and Nichele co-supervised the work. Olsen, Yazidi, Riegler, Halvorsen, and Nichele reviewed the manuscript.

Story and reflections:

This paper consists of the implementation and results of the conceptual method in Paper B.1. The results were very exciting in the beginning, especially the general emulation of logic gates performed by one adaptable spiking neural network. However, they are simple. Therefore, the objective of the introduction of this method is to create possibilities to improve and be a more biological and adaptable alternative to the rigid backward propagation of errors for training artificial neural networks.

4.6 Paper C.1

Towards self-organized control: using neural cellular automata to robustly control a cart-pole agent

By *Alexandre Variengien, Sidney Pontes-Filho, Tom Glover and Stefano Nichele (2021)*

Published in *Innovations in Machine Intelligence (IMI)*

Neural cellular automata (Neural CA) are a recent framework used to model biological phenomena emerging from multicellular organisms. In these systems, artificial neural networks are used as update rules for cellular automata. Neural CA are end-to-end differentiable systems where the parameters of the neural network can be learned to achieve a particular task. In this work, we used neural CA to control a cart-pole agent. The observations of the environment are transmitted in input cells while the values of output cells are used as a readout of the system. We trained the model using deep-Q learning where the states of the output cells were used as the Q-value estimates to be optimized. We found that the computing abilities of the cellular automata were maintained over several hundreds of thousands of iterations, producing an emergent stable behavior in the environment it controls for thousands of steps. Moreover, the system demonstrated life-like phenomena such as a developmental phase, regeneration after damage, stability despite a noisy environment, and robustness to unseen disruption such as input deletion.

Role of the authors:

Pontes-Filho and Nichele discussed the initial idea for the work. Variengien conducted the experimental work and wrote the paper. Pontes-Filho, Nichele, and Glover supervised the work and reviewed the manuscript.

Story and reflections:

The discussion of the idea for this paper happened when Glover presented in our lab the method "Growing Neural Cellular Automata" [32]. The purpose of this idea is to leverage the self-organization of neural cellular automata for control tasks. This method is envisaged to have some sort of plasticity or even meta-plasticity as an emergent behavior to solve some tasks that require such features. In this thesis, this paper is the only one that uses backward propagation of errors as an optimization method for the artificial neural network that defines the rule of the neural cellular automata.

4.7 Paper C.2

A unified substrate for body-brain co-evolution

By *Sidney Pontes-Filho, Kathryn Walker, Elias Najarro, Stefano Nichele and Sebastian Risi (2022)*

Published in *From Cells to Societies: Collective Learning Across Scales at the Tenth International Conference on Learning Representations (ICLR 2022)*

The discovery of complex multicellular organism development took millions of years of evolution. The genome of such a multicellular organism guides the development of its body from a single cell, including its control system. Our goal is to imitate this natural process using a single neural cellular automaton (NCA) as a genome for modular robotic agents. In the introduced approach, called *Neural Cellular Robot Substrate* (NCRS), a single NCA guides the growth of a robot and the cellular activity which controls the robot during deployment. In this paper, NCRSs are trained with covariance matrix adaptation evolution strategy (CMA-ES), and covariance matrix adaptation MAP-Elites (CMA-ME) for quality diversity, which we show leads to more diverse robot morphologies with higher fitness scores. While the NCRS can solve the easier tasks from our benchmark environments, the success rate reduces when the difficulty of the task increases. We discuss directions for future work that may facilitate the use of the NCRS approach for more complex domains.

Role of the authors:

Pontes-Filho had the main idea while discussing with Risi and Najarro. Pontes-Filho implemented the project, ran all the training and simulations, and wrote the paper. Walker and Najarro helped in the implementation. Risi and Nichele supervised the work. Walker, Najarro, Nichele, and Risi contributed through discussions and corrections of the paper.

Story and reflections:

The work in this paper was performed in the Robotics, Evolution and Art Lab at the IT University of Copenhagen. This was a program of the Research Council of Norway for PhD candidates who wish to perform research abroad. The main idea is based on the method in Paper C.1 and my desire to develop an approach that gives the necessary freedom to the optimization algorithm for complexifying and developing new solutions of a virtual creature's morphology and controller. Therefore, it might be the first step towards a solution of open-endedness in embodied agents [77].

4.8 Paper C.3

Collective control of modular soft robots via embodied Spiking Neural Cellular Automata

By *Giorgia Nadizar, Eric Medvet, Stefano Nichele and Sidney Pontes-Filho (2022)*

Published in *From Cells to Societies: Collective Learning Across Scales at the Tenth International Conference on Learning Representations (ICLR 2022)*

Voxel-based Soft Robots (VSRs) are a form of modular soft robots, composed of several deformable cubes, i.e., voxels. Each VSR is thus an ensemble of simple agents, namely the voxels, which must cooperate to give rise to the overall VSR behavior. Within this paradigm, collective intelligence plays a key role in enabling the emerge of coordination, as each voxel is independently controlled, exploiting only the local sensory information together with some knowledge passed from its direct neighbors (distributed or collective control). In this work, we propose a novel form of collective control, influenced by Neural Cellular Automata (NCA) and based on the bio-inspired Spiking Neural Networks: the embodied Spiking NCA (SNCA). We experiment with different variants of SNCA, and find them to be competitive with the state-of-the-art distributed controllers for the task of locomotion. In addition, our findings show significant improvement with respect to the baseline in terms of adaptability to unforeseen environmental changes, which could be a determining factor for physical practicability of VSRs.

Role of the authors:

Nadizar had the main idea during a meeting with Medvet, Nichele, and Pontes-Filho. Nadizar performed the experimental work and wrote the paper. Medvet, Nichele, and Pontes-Filho supervised the work and reviewed the paper.

Story and reflections:

The idea of this paper was to merge the work with voxel-based soft robots from Nadizar and Medvet, adaptive spiking neural networks from Paper B.1, and neural cellular automata for control from Paper C.1. The only adaptation applied in the spiking neural cellular automata is homeostasis, and it already affects positively the distributed controller for unforeseen conditions of the environment. The expectation is that spike-timing-dependent plasticity may also be beneficial.

Chapter 5

Discussion

This chapter consists of a discussion of the research questions and how they are addressed by the results reported in this thesis.

5.1 Critical dynamical systems

Dynamical systems at criticality are scarce in the space of possible dynamics. Accordingly, this issue raises the first research question in this thesis:

Research question 1

How can dynamical systems be optimized towards criticality?

The paper category A is the one that addresses this question and Paper A.1 is the first and successful research result to do so. Criticality was obtained in unidimensional stochastic cellular automata with three neighbors and binary states. The fitness score is calculated by a function, which is the weighted sum of normalized multi-objective scores. The objective of the fitness function is to find avalanche distributions that are power laws. If the objective is reached, the evolved system is considered to have critical behavior. The multi-objective scores are calculated using linear fitting with least squares regression in the avalanche distributions where the size and occurrence rate are mapped to logarithmic scale. In this way, the linear fitting is still useful and faster than power law model fitting with maximum likelihood estimator [78].

During the research and experiments for the resulting paper, it was also verified that linear fitting for power law distribution estimation is more beneficial for being used to calculate the fitness score because the Kolmogorov-Smirnov statistic used to compare the theoretical model with the empirical data of the avalanche distributions gives a much larger error when the distribution is not a power law. Thus, it makes the fitness landscape less flat.

Paper A.2 confirms and improves the findings of its previous work. The fitness function is improved to give a more accurate score for power law distribution similarity while maintaining the efficiency in convergence during the optimization process with genetic algorithm. In this paper, this fitness function managed to guide a stochastic echo state network towards criticality as well. This was applied in another dynamical system, which is the random Boolean network. However, this system was the most difficult one to optimize towards critical behavior.

With the results of Papers A.1 and A.2, research question 1 is successfully addressed. Two of the three stochastic dynamical systems clearly achieved criticality. They are cellular automata, and recurrent neural networks with perceptron neuron model, also known as echo state networks. The latter system is a particular type of artificial neural network that is commonly used in reservoir computing. The stochastic system that requires more investigation to evolve critical behavior is the random Boolean network. The answer to this question is that the goal of the fitness function can be the approximation of the avalanche distributions to a power law. Therefore, during optimization or artificial evolution, it can guide the system to be in a critical state. Paper A.3 goes further in this analysis and tests the robustness of the critical behavior in the optimal stochastic cellular automata found in Paper A.2. Such a verification indicates how a critical system can self-organize and be maintained because these characteristics are part of the hypothesis of how the brain resists noise.

5.2 Intelligent dynamical systems

Dynamical systems have benefits for being used in artificial intelligence, especially because of their capacity to adapt. However, they are rarely applied and are hard to control [12]. Thus, the second research question in this thesis can be formulated:

Research question 2

How can dynamical systems be optimized to present adaptable and intelligent behavior?

There are two paper categories that address this research question. They are paper categories B and C. The first paper for that is Paper B.1. It describes a conceptual framework that co-evolves the topology of a spiking neural network and the neuroplasticity of its neurons. The idea behind the proposed method is that it does not involve the inheritance of synaptic strength during evolution. Therefore, adaptation of the connection weights must be present. This is also forced by the abrupt change of the environment that the agent needs to survive and is controlled by the evolving spiking neural network. In Paper C.2, such a conceptual method is implemented, and its results indicate that a fully evolved spiking neural network is capable of adapting even to unforeseen environment conditions, which is the case for the general emulation of logic gates.

The research related to the neural cellular automata for control tasks in paper category C presents results that show intelligent and adaptable behavior. Paper C.1 is the first one in this category and contains a method that is extremely adaptable and robust for intelligently controlling a cart balancing a pole. The NCA is resistant to perturbations in the states of the cells, and to missing inputs. Paper C.2 introduces a single NCA for sequentially developing and controlling a virtual two-dimensional modular robot consisting of body modules, wheels in the same orientation, and light sensors. Such a robot has the objective of chasing a light bulb or carrying a ball to a target area. In this work, there was no analysis of the existence of adaptable features. However, since it is a dynamical system, adaptability can be present. Paper C.3 presents a spiking neural cellular automaton for distributively controlling voxel-based soft robots in two dimensions. This system has a neuroplasticity that maintains the balance of the firing rate of the neurons, which is homeostasis. This homeostatic plasticity significantly improves the adaptability of the agent's controller. This was verified by assessing the agent in unforeseen terrains.

Research question 2 is addressed by showing that dynamical systems are optimizable, and the methods applied for optimizing them were neuroevolution of augmenting topologies, evolution strategy, deep Q-learning, and evolutionary algorithm. In paper category B, the dynamical system is a spiking neural network, in which artificial evolution optimizes the structure of the network and the plasticity of the neurons. However, the weights of the connections are randomly initialized and are not optimized. Therefore, it is a weight agnostic neural network. In paper category C, the neural cellular automata already have a predefined architecture of the artificial neural networks, and their optimization searches for optimal weights and biases.

5.3 Benefits of self-organization in artificial intelligence

Self-organizing dynamical systems are an alternative to the current AI systems that are rigid and lack adaptability to new environment or data conditions. Since self-organizing systems may solve those issues, it brings the third research question in this thesis:

Research question 3

Are there benefits to using self-organizing dynamical systems in artificial intelligence? If so, what are they?

The same paper categories for research question 2 also address this question. In paper category B, there are the systems with self-organizing weights through local interactions and evolvable topologies that allow for recurrent connections. Paper category C describes systems with local interactions and memory (cell state persistence) that are the neural cellular automata.

The method NAGI is presented in the paper category B. The adaptive spiking neural networks optimized by the modified NEAT solve the simple tasks with random initialization of the connection weights and change in the environment conditions. Such capacity indicates high adaptation. An optimized system can also be successful in environment conditions that were never shown during optimization.

The neural cellular automata for controlling the cart-pole agent in Paper C.1 presents self-organization and it provides the system with adaptable, developmental, and regenerating behaviors. Therefore, the system can suffer damage, though it quickly recovers from that for maintaining the pole still balanced. It can start with a completely random state and even resist input deletion. Moreover, the NCA has an asynchronous and noisy update of the cells that favors long-term stability. Paper C.2 introduces a single NCA for the development and control of a modular robot. The self-organizing system provides a unified substrate to be optimized for morphogenesis from a single cell and distributed control. There was a task where the morphology of the robot was more important than accurate control, and the evolved NCA was successful in producing an adequate robot body. Paper C.3 describes the spiking NCA with an additional layer of self-organization, which is the adaptive threshold of the spiking neurons. As previously mentioned, this dynamical system is adaptable and intelligent.

Regarding this research question, the results confirm that there are benefits to having self-organizing artificial intelligence systems. Some of the main benefits are adaptability, developmental capability, and regeneration as well as robustness to noise, environmental changes, and missing information.

Chapter 6

Concluding remarks

This final chapter of the research overview presents a summary and analysis of the research outcome. Some ideas for advancing the research work in this thesis are also discussed.

6.1 Conclusion

The field of AI is advancing rapidly, but it consists of mostly specialized solutions and computing power. It is estimated that in a few years the growth in computing power will plateau. Therefore, AI in future may need to rely on new algorithmic breakthroughs [9]. In this thesis, a shift from rigid AI systems to more dynamical ones is proposed such that the dynamics of the substrate can support more adaptability and generalization.

The research in this thesis promotes the progression of the application of evolutionary algorithms for criticality, the employment of a deep learning library that takes advantage of the parallelization in graphics cards for the simulation of the simplest to the most complex dynamical systems, the evolution of the topology and plasticity of spiking neural networks that control embodied agents and adapt through sensory feedback learning, and the use of self-organizing systems, such as neural cellular automata, in control tasks.

All these new algorithms serve as the first step towards alternatives to the most common AI innovations in recent years. They bring the benefits of dynamical systems to AI, as observed from the results presented in this thesis. Examples of advantages that dynamical AI systems may provide are robustness, the ability to regenerate when damaged, and application in the design and control of modular robots. They may also be more energy efficient in the context of reservoir computing, for example with spiking neural networks executed in neuromorphic computers. Those innovations might be useful steps towards artificial general intelligence and open-endedness.

6.2 Contributions

Besides addressing the research questions as discussed in Chapter 5, the results in this thesis consist of several self-organizing artificial intelligence systems, and the application of genetic algorithm for criticality, which may increase the computational capacity of a dynamical system.

The paper category A provides a Python library based on TensorFlow for the simulation and evolution of different dynamical systems, from cellular automata to liquid state machines (spiking neural networks). This library is called EvoDynamic[1]. The paper category B introduces a framework for low-level artificial general intelligence, named NAGI[2]. The three papers in paper category C individually contribute with a different method each, namely self-organized control[3] from Paper C.1, the neural cellular robot substrate (NCRS)[4] from Paper C.2, and the spiking NCA for collective control of voxel-based soft robots[5] from Paper C.3.

[1]EvoDynamic open-source repository at `https://github.com/SocratesNFR/EvoDynamic`

[2]NAGI open-source repository at `https://github.com/SocratesNFR/neat-nagi-python`

[3]Interactive preprint of self-organized control at `https://avariengien.github.io/self-organized-control/`

[4]NCRS open-source repository at `https://github.com/sidneyp/neural-cellular-robot-substrate`

[5]Spiking NCA open-source repository at `https://github.com/giorgia-nadizar/VSRCollectiveControlViaSNCA`

6.3 Future work

The natural next steps for the research line in the paper category A are the application of the optimized critical systems in reservoir computing. In this way, the computational capacity of such systems can be analyzed, and their performance correlated with criticality. Another idea to advance the research in Paper A.3 is to verify the robustness of the stochastic cellular automata with criticality as a reservoir.

For NAGI from paper category B, the plan is to increase the complexity of the tasks. For example, the classification tasks would have more classes and more dimensions in the input data. In case of failure to advance to more complex tasks, some constraints in NAGI can be removed, such as allowing neuroplasticity to be defined for each connection instead of for each neuron. This would increase the dimensionality of the search space, but it was proven to work for the method of Najarro and Risi [79].

The three methods with neural cellular automata for control described in paper category C have their specific future works. Nevertheless, in general, the plans to advance those works are to apply NCA in more complex control tasks. In case there is any issue with training the NCAs for a higher difficulty, the idea is to train the NCA by slowly increasing the difficulty of the task as in curriculum learning [80]. If the plan requires a wide grid for the NCA, the strategy may be to start with a small grid that grows over time, such as in the variational neural cellular automata [81]. An ambition for this approach is that the NCA could show emergence of the dynamical features of the brain, such as meta-learning [82] or even metaplasticity [83]. This could be achieved by designing tasks that require such intelligent dynamics.

Bibliography

[1] S. Pontes-Filho, P. Lind, A. Yazidi, J. Zhang, H. Hammer, G. Mello, I. Sandvig, G. Tufte, and S. Nichele, "Evodynamic: A framework for the evolution of generally represented dynamical systems and its application to criticality," in *International Conference on the Applications of Evolutionary Computation (Part of EvoStar)*. Springer, 2020, pp. 133–148.

[2] ——, "A neuro-inspired general framework for the evolution of stochastic dynamical systems: Cellular automata, random boolean networks and echo state networks towards criticality," *Cognitive Neurodynamics*, vol. 14, no. 5, pp. 657–674, 2020.

[3] S. Pontes-Filho, P. G. Lind, and S. Nichele, "Assessing the robustness of critical behavior in stochastic cellular automata," *Physica D: Nonlinear Phenomena*, vol. 441, p. 133507, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167278922002135

[4] S. Pontes-Filho and S. Nichele, "A conceptual bio-inspired framework for the evolution of artificial general intelligence," *arXiv preprint arXiv:1903.10410*, 2019.

[5] S. Pontes-Filho, K. Olsen, A. Yazidi, M. A. Riegler, P. Halvorsen, and S. Nichele, "Towards the neuroevolution of low-level artificial general intelligence," *Frontiers in Robotics and AI*, vol. 9, 2022. [Online]. Available: https://www.frontiersin.org/articles/10.3389/frobt.2022.1007547

[6] A. Variengien, S. Pontes-Filho, T. Glover, and S. Nichele, "Towards self-organized control: Using neural cellular automata to robustly control a cart-pole agent," *Innovations in Machine Intelligence (IMI)*, vol. 1, pp. 1–14, 2021.

[7] S. Pontes-Filho, K. Walker, E. Najarro, S. Nichele, and S. Risi, "A unified substrate for body-brain co-evolution," in *From Cells to Societies: Collective Learning across Scales*, 2022.

[8] G. Nadizar, E. Medvet, S. Nichele, and S. Pontes-Filho, "Collective control of modular soft robots via embodied spiking neural cellular automata," in *From Cells to Societies: Collective Learning across Scales*, 2022.

[9] A. Lohn and M. Musser, “AI and compute: How much longer can computing power drive artificial intelligence progress?” Center for Security and Emerging Technology, Tech. Rep., Jan. 2022. [Online]. Available: https://doi.org/10.51593/2021ca009

[10] K. O. Stanley, J. Clune, J. Lehman, and R. Miikkulainen, “Designing neural networks through neuroevolution,” *Nature Machine Intelligence*, vol. 1, no. 1, pp. 24–35, 2019.

[11] M. Mitchell, “Why AI is harder than we think,” *arXiv preprint arXiv:2104.12871*, 2021.

[12] S. Risi, “The future of artificial intelligence is self-organizing and self-assembling,” *sebastianrisi.com*, 2021. [Online]. Available: https://sebastianrisi.com/self_assembling_ai

[13] J. McCarthy, M. L. Minsky, N. Rochester, and C. E. Shannon, “A proposal for the Dartmouth summer research project on artificial intelligence, august 31, 1955,” *AI Magazine*, vol. 27, no. 4, pp. 12–12, 2006.

[14] S. Dick, “Artificial Intelligence,” *Harvard Data Science Review*, vol. 1, no. 1, jul 1 2019, https://hdsr.mitpress.mit.edu/pub/0aytgrau.

[15] L. Squire, D. Berg, F. E. Bloom, S. Du Lac, A. Ghosh, and N. C. Spitzer, *Fundamental neuroscience*. Academic Press, 2012.

[16] H. Sayama, *Introduction to the modeling and analysis of complex systems*. Open SUNY Textbooks, 2015.

[17] W. Tang and Q. Wu, “Evolutionary computation,” in *Condition monitoring and assessment of power transformers using computational intelligence*. Springer, 2011, pp. 15–36.

[18] C. G. Langton, *Artificial life: An overview*. MIT Press, 1997.

[19] K. Heiney, O. Huse Ramstad, V. Fiskum, N. Christiansen, A. Sandvig, S. Nichele, and I. Sandvig, “Criticality, connectivity, and neural disorder: A multifaceted approach to neural computation,” *Frontiers in Computational Neuroscience*, vol. 15, p. 611183, 2021.

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

[21] C. Gershenson, “Introduction to random Boolean networks,” *arXiv preprint nlin/0408006*, 2004.

[22] H. Jaeger, "The "echo state" approach to analysing and training recurrent neural networks-with an erratum note," *Bonn, Germany: German National Research Center for Information Technology GMD Technical Report*, vol. 148, no. 34, p. 13, 2001.

[23] H. Jaeger and H. Haas, "Harnessing nonlinearity: Predicting chaotic systems and saving energy in wireless communication," *Science*, vol. 304, no. 5667, pp. 78–80, 2004. [Online]. Available: https://science.sciencemag.org/content/304/5667/78

[24] W. Maass, T. Natschläger, and H. Markram, "Real-time computing without stable states: A new framework for neural computation based on perturbations," *Neural Computation*, vol. 14, no. 11, pp. 2531–2560, 2002.

[25] W. Maass and H. Markram, "On the computational power of circuits of spiking neurons," *Journal of Computer and System Sciences*, vol. 69, no. 4, pp. 593 – 616, 2004. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0022000004000406

[26] J. H. Holland, "Genetic algorithms," *Scientific American*, vol. 267, no. 1, pp. 66–73, 1992.

[27] N. Hansen and A. Ostermeier, "Adapting arbitrary normal mutation distributions in evolution strategies: The covariance matrix adaptation," in *Proceedings of IEEE international conference on evolutionary computation*. IEEE, 1996, pp. 312–317.

[28] J.-B. Mouret and J. Clune, "Illuminating search spaces by mapping elites," *arXiv preprint arXiv:1504.04909*, 2015.

[29] T. Hester, M. Vecerik, O. Pietquin, M. Lanctot, T. Schaul, B. Piot, D. Horgan, J. Quan, A. Sendonaris, I. Osband *et al.*, "Deep Q-learning from demonstrations," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, no. 1, 2018.

[30] X. Li and A. G.-O. Yeh, "Neural-network-based cellular automata for simulating multiple land use changes using GIS," *International Journal of Geographical Information Science*, vol. 16, no. 4, pp. 323–343, 2002.

[31] S. Nichele, M. B. Ose, S. Risi, and G. Tufte, "CA-NEAT: Evolved compositional pattern producing networks for cellular automata morphogenesis and replication," *IEEE Transactions on Cognitive and Developmental Systems*, vol. 10, no. 3, pp. 687–700, 2017.

[32] A. Mordvintsev, E. Randazzo, E. Niklasson, and M. Levin, "Growing neural cellular automata," *Distill*, vol. 5, no. 2, p. e23, 2020.

[33] L. Brochini, A. de Andrade Costa, M. Abadi, A. C. Roque, J. Stolfi, and O. Kinouchi, "Phase transitions and self-organized criticality in networks of stochastic spiking neurons," *Scientific reports*, vol. 6, no. 1, pp. 1–15, 2016.

[34] C. G. Langton, "Computation at the edge of chaos: Phase transitions and emergent computation," *Physica D: Nonlinear Phenomena*, vol. 42, no. 1-3, pp. 12–37, 1990.

[35] P. Bak, C. Tang, and K. Wiesenfeld, "Self-organized criticality: An explanation of the 1/f noise," *Physical review letters*, vol. 59, no. 4, p. 381, 1987.

[36] V. Pasquale, P. Massobrio, L. Bologna, M. Chiappalone, and S. Martinoia, "Self-organization and neuronal avalanches in networks of dissociated cortical neurons," *Neuroscience*, vol. 153, no. 4, pp. 1354–1369, 2008.

[37] M. Costandi, *Neuroplasticity*. MIT Press, 2016.

[38] R. Pfeifer and C. Scheier, *Understanding intelligence*. MIT Press, 2001.

[39] P. Wang, "On defining artificial intelligence," *Journal of Artificial General Intelligence*, vol. 10, no. 2, pp. 1–37, 2019. [Online]. Available: https://doi.org/10.2478/jagi-2019-0002

[40] M. Minsky, *Society of mind*. Simon and Schuster, 1988.

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

[42] G. F. Luger, *Artificial intelligence: Structures and strategies for complex problem solving*. Pearson education, 2005.

[43] F. Rosenblatt, "The perceptron: A probabilistic model for information storage and organization in the brain." *Psychological review*, vol. 65, no. 6, p. 386, 1958.

[44] S. Marsland, *Machine learning: An algorithmic perspective*. Chapman and Hall/CRC, 2011.

[45] D. R. Nayak, A. Mahapatra, and P. Mishra, "A survey on rainfall prediction using artificial neural network," *International journal of computer applications*, vol. 72, no. 16, 2013.

[46] M. L. Jones, "How we became instrumentalists (again) data positivism since World War II," *Historical Studies in the Natural Sciences*, vol. 48, no. 5, pp. 673–684, 2018.

[47] L. Noriega, "Multilayer perceptron tutorial," *School of Computing. Staffordshire University*, 2005.

[48] L. A. Camuñas-Mesa, B. Linares-Barranco, and T. Serrano-Gotarredona, "Neuromorphic spiking neural networks and their memristor-CMOS hardware implementations," *Materials*, vol. 12, no. 17, 2019. [Online]. Available: https://www.mdpi.com/1996-1944/12/17/2745

[49] A. Tavanaei, M. Ghodrati, S. R. Kheradpisheh, T. Masquelier, and A. Maida, "Deep learning in spiking neural networks," *Neural Networks*, vol. 111, pp. 47–63, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0893608018303332

[50] B. Schrauwen, D. Verstraeten, and J. Van Campenhout, "An overview of reservoir computing: Theory, applications and implementations," in *Proceedings of the 15th European symposium on artificial neural networks. p. 471-482 2007*, 2007, pp. 471–482.

[51] G. Tanaka, T. Yamane, J. B. Héroux, R. Nakane, N. Kanazawa, S. Takeda, H. Numata, D. Nakano, and A. Hirose, "Recent advances in physical reservoir computing: A review," *Neural Networks*, vol. 115, pp. 100–123, 2019.

[52] *Computation in artificial spin ice*, ser. ALIFE 2022: The 2022 Conference on Artificial Life. MIT Press, 07 2018. [Online]. Available: https://doi.org/10.1162/isal_a_00011

[53] P. Wang and B. Goertzel, "Introduction: Aspects of artificial general intelligence," in *Proceedings of the 2007 conference on Advances in Artificial General Intelligence: Concepts, Architectures and Algorithms: Proceedings of the AGI Workshop 2006*, 2007, pp. 1–16.

[54] K. Mainzer, *Thinking in complexity: The computational dynamics of matter, mind, and mankind.* Springer Science & Business Media, 2007.

[55] S. Nichele, "Evolvability, complexity and scalability of cellular evolutionary and developmental systems," Ph.D. dissertation, NTNU, 2015.

[56] P. Topa, "Network systems modelled by complex cellular automata paradigm," *Cellular Automata-Simplicity behind Complexity*, pp. 259–274, 2011.

[57] J. v. Neumann, "Theory of self-reproducing automata," *Edited by Arthur W. Burks*, 1966.

[58] D. V. Buonomano and M. M. Merzenich, "Temporal information transformed into a spatial code by a neural network with realistic properties," *Science*, vol. 267, no. 5200, pp. 1028–1030, 1995.

[59] G. Dion, A. I.-E. Oudrhiri, B. Barazani, A. Tessier-Poirier, and J. Sylvestre, "Reservoir computing in mems," in *Reservoir Computing*. Springer, 2021, pp. 191–217.

[60] S. Wolfram, "Universality and complexity in cellular automata," *Physica D: Nonlinear Phenomena*, vol. 10, no. 1-2, pp. 1–35, 1984.

[61] J. Alstott, E. Bullmore, and D. Plenz, "powerlaw: A python package for analysis of heavy-tailed distributions," *PloS one*, vol. 9, no. 1, p. e85777, 2014.

[62] K. Christensen and N. R. Moloney, *Complexity and criticality*. World Scientific Publishing Company, 2005, vol. 1.

[63] A. M. Zador, "A critique of pure learning and what artificial neural networks can learn from animal brains," *Nature communications*, vol. 10, no. 1, pp. 1–7, 2019.

[64] G. Tufte, "Phenotypic, developmental and computational resources: Scaling in artificial development," in *Proceedings of the 10th Annual Conference on Genetic and Evolutionary Computation*, 2008, pp. 859–866.

[65] R. Mercado, V. Muñoz-Jiménez, M. Ramos, and F. Ramos, "Generation of virtual creatures under multidisciplinary biological premises," *Artificial Life and Robotics*, vol. 27, no. 3, pp. 495–505, 2022.

[66] R. Doursat, H. Sayama, and O. Michel, "A review of morphogenetic engineering," *Natural Computing*, vol. 12, no. 4, pp. 517–535, 2013.

[67] K. O. Stanley and R. Miikkulainen, "A taxonomy for artificial embryogeny," *Artificial Life*, vol. 9, no. 2, pp. 93–130, 2003.

[68] S. Sudhakaran, D. Grbic, S. Li, A. Katona, E. Najarro, C. Glanois, and S. Risi, "Growing 3d artefacts and functional machines with neural cellular automata," *arXiv preprint arXiv:2103.08737*, 2021.

[69] C. Darwin, *On the Origin of Species by Means of Natural Selection*. London: Murray, 1859.

[70] A. E. Eiben, J. E. Smith *et al.*, *Introduction to evolutionary computing*. Springer, 2003, vol. 53.

[71] W. Banzhaf, "Evolutionary computation and genetic programming," in *Engineered Biomimicry*, A. Lakhtakia and R. J. Martín-Palma, Eds. Boston: Elsevier, 2013, pp. 429–447. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780124159952000179

[72] B. L. Hogan, "Morphogenesis," *Cell*, vol. 96, no. 2, pp. 225–233, 1999.

[73] S. Pontes-Filho, A. Yazidi, J. Zhang, H. Hammer, G. Mello, I. Sandvig, G. Tufte, and S. Nichele, "A general representation of dynamical systems for reservoir computing," in *Workshop on Novel Substrates and Models for the Emergence of Developmental, Learning and Cognitive Capabilities at IEEE ICDL-EPIROB 2019*, 2019.

[74] K. Heiney, O. H. Ramstad, S. Pontes-Filho, T. Glover, T. Lindell, J. J. Farner, H. Weydahl, and S. Nichele, "Bridging the computational gap: From biological to artificial substrates," in *Second International Workshop on Theoretical and Experimental Material Computing (TEMC 2020), Artificial Life Conference (ALife 2021)*, 2021.

[75] G. B. M. e Mello, S. Pontes-Filho, I. Sandvig, V. D. Valderhaug, E. Zouganeli, O. H. Ramstad, A. Sandvig, and S. Nichele, "Method to obtain neuromorphic reservoir networks from images of in vitro cortical networks," in *2019 IEEE Symposium Series on Computational Intelligence (SSCI)*. IEEE, 2019, pp. 2360–2366.

[76] S. Pontes-Filho, A. G. Dahl, S. Nichele, and G. B. M. e Mello, "A deep learning-based tool for automatic brain extraction from functional magnetic resonance images of rodents," in *Proceedings of SAI Intelligent Systems Conference*. Springer, 2021, pp. 549–558.

[77] K. O. Stanley, "Why open-endedness matters," *Artificial Life*, vol. 25, no. 3, pp. 232–235, 2019.

[78] A. Clauset, C. R. Shalizi, and M. E. Newman, "Power-law distributions in empirical data," *SIAM review*, vol. 51, no. 4, pp. 661–703, 2009.

[79] E. Najarro and S. Risi, "Meta-learning through hebbian plasticity in random networks," *Advances in Neural Information Processing Systems*, vol. 33, pp. 20 719–20 731, 2020.

[80] Y. Bengio, J. Louradour, R. Collobert, and J. Weston, "Curriculum learning," in *Proceedings of the 26th Annual International Conference on Machine Learning*, 2009, pp. 41–48.

[81] R. B. Palm, M. González-Duque, S. Sudhakaran, and S. Risi, "Variational neural cellular automata," *arXiv preprint arXiv:2201.12360*, 2022.

[82] J. X. Wang, "Meta-learning in natural and artificial intelligence," *Current Opinion in Behavioral Sciences*, vol. 38, pp. 90–95, 2021.

[83] W. C. Abraham, "Metaplasticity: Tuning synapses and networks for plasticity," *Nature Reviews Neuroscience*, vol. 9, no. 5, pp. 387–387, 2008.

Part II

Publications

Paper A.1

EvoDynamic: a framework for the evolution of generally represented dynamical systems and its application to criticality

(Pontes-Filho et al., 2020)

Author(s):

Sidney Pontes-Filho, Pedro Lind, Anis Yazidi, Jianhua Zhang, Hugo Hammer, Gustavo B. M. Mello, Ioanna Sandvig, Gunnar Tufte and Stefano Nichele

Published at conference:

EvoApplications 2020: International Conference on the Applications of Evolutionary Computation (Part of EvoStar)

Paper A.1
Pontes-Filho et al. (2020)

EvoDynamic: A Framework for the Evolution of Generally Represented Dynamical Systems and Its Application to Criticality

Sidney Pontes-Filho[1,2], Pedro Lind[1], Anis Yazidi[1], Jianhua Zhang[1], Hugo Hammer[1], Gustavo B. M. Mello[1], Ioanna Sandvig[3], Gunnar Tufte[2], and Stefano Nichele[1,4]

[1] Department of Computer Science, Oslo Metropolitan University, Oslo, Norway
sidneyp@oslomet.no
[2] Department of Computer Science, Norwegian University of Science and Technology, Trondheim, Norway
[3] Department of Neuromedicine and Movement Science, Norwegian University of Science and Technology, Trondheim, Norway
[4] Holistic Systems, SimulaMet, Oslo, Norway

Abstract. Dynamical systems possess a computational capacity that may be exploited in a reservoir computing paradigm. This paper presents a general representation of dynamical systems which is based on matrix multiplication. That is similar to how an artificial neural network (ANN) is represented in a deep learning library and its computation can be faster because of the optimized matrix operations that such type of libraries have. Initially, we implement the simplest dynamical system, a cellular automaton. The mathematical fundamentals behind an ANN are maintained, but the weights of the connections and the activation function are adjusted to work as an update rule in the context of cellular automata. The advantages of such implementation are its usage on specialized and optimized deep learning libraries, the capabilities to generalize it to other types of networks and the possibility to evolve cellular automata and other dynamical systems in terms of connectivity, update and learning rules. Our implementation of cellular automata constitutes an initial step towards a more general framework for dynamical systems. Our objective is to evolve such systems to optimize their usage in reservoir computing and to model physical computing substrates. Furthermore, we present promising preliminary results toward the evolution of complex behavior and criticality using genetic algorithm in stochastic elementary cellular automata.

Keywords: Cellular automata · Dynamical systems · Implementation · Reservoir computing · Evolution · Criticality

P. A. Castillo et al. (Eds.): EvoApplications 2020, LNCS 12104, pp. 133–148, 2020.
https://doi.org/10.1007/978-3-030-43722-0_9

1 Introduction

A cellular automaton (CA) is the simplest computing system where the emergence of complex dynamics from local interactions might take place. It consists of a grid of cells with a finite number of states that change according to simple rules depending on the neighborhood and own state in discrete time-steps. Some notable examples are the elementary CA [30], which is unidimensional with three neighbors and eight update cases, and Conway's Game of Life [24], which is two-dimensional with nine neighbors and three update cases.

Table 1 presents some computing systems that are capable of giving rise to the emergence of complex dynamics. Those systems can be exploited by reservoir computing, which is a paradigm that resorts to dynamical systems to simplify complex data. Such simplification means that reservoir computing utilizes the non-linear dynamical system to perform a non-linear transformation from non-linear data to higher dimensional linear data. Such linearized data can be applied in linear machine learning methods which are faster for training and computing because has less trainable variables and operations. Hence, reservoir computing is more energy efficient than deep learning methods and it can even yield competitive results, especially for temporal data [25,27]. Basically, reservoir computing exploits a dynamical system that possesses the echo state property and fading memory, where the internals of the reservoir are untrained and the only training happens at the linear readout stage [16].

Reservoir computers are most useful when the substrate's dynamics are at the "edge of chaos" [17], meaning a range of dynamical behaviors that is between order and disorder. Cellular automata with such dynamical behavior are capable of being exploited as reservoirs [21,22]. Other systems can also exhibit similar dynamics. The coupled map lattice [15] is very similar to CA, the only exception is that the coupled map lattice has continuous states which are updated by a recurrence equation involving the neighborhood. Random Boolean network [10] is a generalization of CA where random connectivity exists. Echo state network [13] is an artificial neural network (ANN) with random topology while liquid state machine [18] is similar to echo state network with the difference that it is a spiking neural network that communicates through discrete-events (spikes) over continuous time.

Table 1. Examples of dynamical systems.

Dynamical system	State	Time	Connectivity
Cellular automata	Discrete	Discrete	Regular
Coupled map lattice	Continuous	Discrete	Regular
Random Boolean network	Discrete	Discrete	Random
Echo state network	Continuous	Discrete	Random
Liquid state machine	Discrete	Continuous	Random

One important aspect of the computation performed in a dynamical system is the trajectory of system states traversed during the computation [19]. Such trajectory may be guided by system parameters [23]. Another characteristic of a dynamical system, which is crucial for computation, is to be in a critical state, as indicated by Langton [17]. If the attractors of the system are in the critical state, this characteristic is called self-organized criticality [7].

Besides, computation in dynamical systems may be carried out in physical substrates [27], such as networks of biological neurons [3] or in nanoscale materials [8]. Finding the correct abstraction for the computation in a dynamical system, e.g. CA, is an open problem [20].

All the systems described in Table 1 are sparsely connected and can be represented by a weighted adjacency matrix, such as a graph. The connectivity from a layer to another in a fully connected feedforward ANN is represented with a weighted adjacency matrix that contains the weights of each connection. Our CA implementation is similar to this, but the connectivity goes from the "layer" of cells to itself.

The goal of representing CA with a weighted adjacency matrix is to implement a framework which facilitates the development of all types of CAs, from unidimensional to multidimensional, with all kinds of lattices and without any boundary conditions during execution; and also allowing the inclusion of other major dynamical systems, independent of the type of the state, time and connectivity. Such initial implementation is the first component of a Python framework under development, based on TensorFlow deep neural network library [4]. Therefore, it benefits from powerful and parallel computing systems with multi-CPU and multi-GPU. One of the framework's goals is to have a balance between performance and generalization of computing dynamical systems, since general methods are slower than specialized ones. Nevertheless, this framework, called EvoDynamic[1], aims at evolving (i.e., using evolutionary algorithms) the connectivity, update and learning rules of sparsely connected networks to improve their usage for reservoir computing guided by the echo state property, fading memory, state trajectory, and other quality measurements. Such improvement of reservoirs is applied similarly in [26], where the internal connectivity of a reservoir is trained to increase its performance to several tasks. Moreover, evolution will model the dynamics and behavior of physical reservoirs, such as *in-vitro* biological neural networks interfaced with microelectrode arrays, and nanomagnetic ensembles. Those two substrates have real applicability as reservoirs. For example, the former substrate is applied to control a robot, in fact making it into a cyborg, a closed-loop biological-artificial neuro-system [3], and the latter possesses computation capability as shown by a square lattice of nanomagnets [14]. Those substrates are the main interest of the SOCRATES project [1] which aims to explore a dynamic, robust and energy efficient hardware for data analysis.

There exist some implementations of CA similar to the one of EvoDynamic framework. They typically implement Conway's Game of Life by applying 2D

[1] EvoDynamic open-source repository is available at https://github.com/SocratesNFR/EvoDynamic.

convolution with a kernel that is used to count the “alive” neighbors, then the resulting matrix consists of the number of “alive” neighboring cells and is used to update the CA. One such implementation, also based on TensorFlow, is available open-source in [2]. We do not use this type of method because it works only with a regular grid topology and not with a random or custom one. Therefore, that cannot be a general method for simulating the different types of dynamical systems.

This paper is organized as follows. Section 2 describes our method according to which we use weighted adjacency matrix to compute CA. Section 3 presents the results obtained from the method. Section 4 discusses the initial advances and future plan of EvoDynamic framework and Sect. 5 concludes this paper.

2 Method

In our proposed method, the equation to calculate the next states of the cells in a cellular automaton is

$$\mathbf{c}_{t+1} = f(\mathbf{A} \cdot \mathbf{c}_t). \tag{1}$$

It is similar to the equation of the forward pass of an artificial neural network, but without the bias. The layer is connected to itself, and the activation function f defines the update rules of the CA. The next states of the CA $\mathbf{c}_{t+1}$ is calculated from the result of the activation function f which receives as argument the dot product between the weighted adjacency matrix $\mathbf{A}$ and the current states of the CA $\mathbf{c}_t$. $\mathbf{c}$ is always a column vector of size $len(\mathbf{c}) \times 1$, that does not depend on how many dimensions the CA has, and $\mathbf{A}$ is a matrix of size $len(\mathbf{c}) \times len(\mathbf{c})$. Hence the result of $\mathbf{A} \cdot \mathbf{c}$ is also a column vector of size $len(\mathbf{c}) \times 1$ as $\mathbf{c}$.

The implementation of cellular automata as an artificial neural network requires the procedural generation of the weighted adjacency matrix of the grid. In this way, any lattice type or multidimensional CAs can be implemented using the same approach. The adjacency matrix of a sparsely connected network contains many zeros because of the small number of connections. Since we implement it on TensorFlow, the data type of the adjacency matrix is preferably a `SparseTensor`. A dot product with this data type can be up to 9 times faster than the dense counterpart. However, it depends on the configuration of the tensors (or, in our case, the adjacency matrices) [28]. The update rule of the CA alters the weights of the connections in the adjacency matrix. In a CA whose cells have two states meaning “dead” (zero) or “alive” (one), the weights in the adjacency matrix are one for connection and zero for no connection, such as an ordinary adjacency matrix. Such matrix facilitates the description of the update rule for counting the number of “alive” neighbors because the result of the dot product between the adjacency matrix and the cell state vector is the vector that contains the number of “alive” neighbors for each cell. If the pattern of the neighborhood matters in the update rule, each cell has its neighbors encoded as a n-ary string where n means the number of states that a cell can have. In this case, the weights of the connections with the neighbors are n-base identifiers and

Algorithm 1. Generation of weighted adjacency matrix for 1D cellular automaton

```
1: procedure GENERATECA1D
2:     $numberOfCells \leftarrow widthCA$
3:     $\mathbf{A} \leftarrow \mathbf{0}^{numberOfCells \times numberOfCells}$     ▷ Adjacency matrix initialization
4:     for $i \leftarrow \{0..(numberOfCells - 1)\}$ do
5:         for $j \leftarrow \{-indexNeighborCenter..(len(neighborhood) - indexNeighborCenter - 1)\}$ do
6:             $currentNeighbor \leftarrow neighborhood_{j+indexNeighborCenter}$
7:             if $currentNeighbor \neq 0 \wedge (isWrappedGrid \vee (\neg isWrappedGrid \wedge (0 \leq (i+j) < widthCA))$ then
8:                 $\mathbf{A}_{i,((i+j) \bmod widthCA)} \leftarrow currentNeighbor$
9:     return A
```

are calculated by

$$neighbor_i = n^i, \forall i \in \{0..len(\mathbf{neighbors}) - 1\} \quad (2)$$

where **neighbors** is a vector of the cell's neighbors. In the adjacency matrix, each neighbor receives a weight according to (2). The result of the dot product with such weighted adjacency matrix is a vector that consists of unique integers per neighborhood pattern. Thus, the activation function is a lookup table from integer (i.e., pattern) to next state.

Algorithm 1 generates the weighted adjacency matrix for one-dimensional CA, such as the elementary CA, where $widthCA$ is the width or number of cells of a unidimensional CA and **neighborhood** is a vector which describes the region around the center cell. The connection weights depend on the type of update rule as previously explained. For example, in case of an elementary CA **neighborhood** $= [4\ 2\ 1]$. $indexNeighborCenter$ is the index of the center cell in the **neighborhood** whose starting index is zero. $isWrappedGrid$ is a Boolean value that works as a flag for adding a wrapped grid or not. A wrapped grid for one-dimensional CA means that the initial and final cells are neighbors. With all these parameters, Algorithm 1 creates an adjacency matrix by looping over the indices of the cells (from zero to $numberOfCells - 1$) with an inner loop for the indices of the neighbors. If the selected $currentNeighbor$ is a non-zero value and its indices do not affect the boundary condition, then the value of $currentNeighbor$ is assigned to the adjacency matrix **A** in the indices that correspond to the connection between the current cell in the outer loop and the actual index of $currentNeighbor$. Finally, this procedure returns the adjacency matrix **A**.

To procedurally generate an adjacency matrix for 2D CA instead of 1D CA, the algorithm needs to have small adjustments. Algorithm 2 shows that for two-dimensional CA, such as Conway's Game of Life. In this case, the height of the CA is an argument passed as $heightCA$. **Neighborhood** is a 2D matrix and **indexNeighborCenter** is a vector of two components meaning the indices of the center of **Neighborhood**. This procedure is similar to the one in Algorithm 1, but it contains one more loop for the additional dimension.

Algorithm 2. Generation of adjacency matrix of 2D cellular automaton

```
 1: procedure GENERATECA2D
 2:     $numberOfCells \leftarrow widthCA * heightCA$
 3:     $\mathbf{A} \leftarrow \mathbf{0}^{numberOfCells \times numberOfCells}$      ▷ Adjacency matrix initialization
 4:     $widthNB, heightNB \leftarrow shape(\mathbf{Neighborhood})$
 5:     for $i \leftarrow \{0..(numberOfCells - 1)\}$ do
 6:         for $j \leftarrow \{-\mathbf{indexNeighborCenter}_0..(widthNB - \mathbf{indexNeighborCenter}_0 - 1)\}$ do
 7:             for $k \leftarrow \{-\mathbf{indexNeighborCenter}_1..(heightNB - \mathbf{indexNeighborCenter}_1 - 1)\}$ do
 8:                 $currentNeighbor \leftarrow Neighborhood_{j+indexNeighborCenter}$
 9:                 if $currentNeighbor \neq 0 \wedge (isWrappedGrid \vee (\neg isWrappedGrid \wedge (0 \leq ((i \bmod heightCA)+j) < widthCA) \wedge (0 \leq (\lfloor i/widthCA \rfloor + k) < heightCA))$ then
10:                     $\mathbf{A}_{i,(((i+k) \bmod widthCA)+((\lfloor i/widthCA \rfloor+j) \bmod heightCA)*widthCA)} \leftarrow currentNeighbor$
11:     return A
```

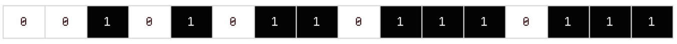
The activation function for CA is different from the ones used for ANN. For CA, it contains the update rules that verify the vector returned by the dot product between the weighted adjacency matrix and the vector of states. Normally, the update rules of the CA are implemented as a lookup table from neighborhood to next state. In our implementation, the lookup table maps the resulting vector of the dot product to the next state of the central cell.

3 Results

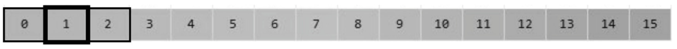
This section presents the results of the proposed method and it also stands for the preliminary results of the EvoDynamic framework.

Figure 1 illustrates a wrapped elementary CA described in the procedure of Algorithm 1 and its generated weighted adjacency matrix. Figure 1a shows the appearance of the desired elementary CA with 16 cells (i.e., $widthCA = 16$). Figure 1b describes its pattern 3-neighborhood and the indices of the cells. Figure 1c shows the result of the Algorithm 1 with the neighborhood calculated by (2) for pattern matching in the activation function. In Fig. 1c, we can verify that the left neighbor has weight equal to 4 (or 2^2 for the most significant bit), central cell weight is 2 (or 2^1) and right neighbor weight is 1 (or 2^0 for the least significant bit) as defined by (2). Since the CA is wrapped, we can notice in row index 0 of the adjacency matrix in Fig. 1c that the left neighbor of cell 0 is the cell 15, and in row index 15 that the right neighbor of cell 15 is the cell 0.

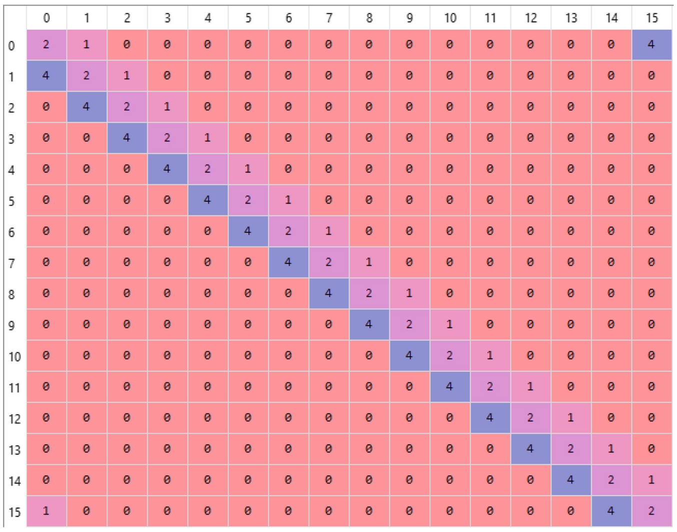
Figure 2 describes a wrapped 2D CA (similar to Game of Life but with less number of neighbors) for Algorithm 2 and shows the resulting adjacency



(a)



(b)

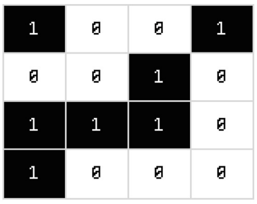


	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	2	1	0	0	0	0	0	0	0	0	0	0	0	0	0	4
1	4	2	1	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	4	2	1	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	4	2	1	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	4	2	1	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	4	2	1	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	4	2	1	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	4	2	1	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	4	2	1	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	4	2	1	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	4	2	1	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	4	2	1	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0	4	2	1	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	4	2	1	0
14	0	0	0	0	0	0	0	0	0	0	0	0	0	4	2	1
15	1	0	0	0	0	0	0	0	0	0	0	0	0	0	4	2

(c)

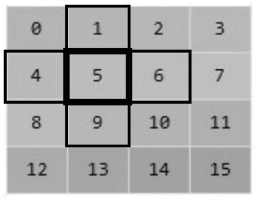
Fig. 1. Elementary cellular automaton with 16 cells and wrapped grid. (a) Example of the grid of cells with states. (b) Indices of the cells and standard pattern neighborhood of elementary CA where thick border means the central cell and thin border means the neighbors. (c) Generated weighted adjacency matrix for the described elementary CA.

matrix. Figure 2a illustrates the desired two-dimensional CA with 16 cells (i.e., $widthCA = 4$ and $heightCA = 4$). Figure 2b presents the von Neumann neighborhood [29] which is used for counting the number of "alive" neighbors (the connection weights are only zero and one, and **Neighborhood** argument of Algorithm 2 defines it). It also shows the index distribution of the CA whose order is preserved after flatting it to a column vector. Figure 2c contains the generated adjacency matrix of Algorithm 2 for the described 2D CA. Figure 2b shows an example of a central cell with its neighbors, the index of this central cell is 5 and the row index 5 in the adjacency matrix of Fig. 2c presents the same neighbor indices, i.e., 1, 4, 6 and 9. Since this is a symmetric matrix, the columns have the same connectivity of the rows. That means the neighborhood of a cell considers this cell as a neighbor too. Therefore, the connections are bidirectional and the adjacency matrix represents an undirected graph. The wrapping effect is also observable. For example, the neighbors of the cell index 0 are 1, 3, 4 and 12. So the neighbors 3 and 12 are the ones that the wrapped grid allowed to exist for cell index 0.



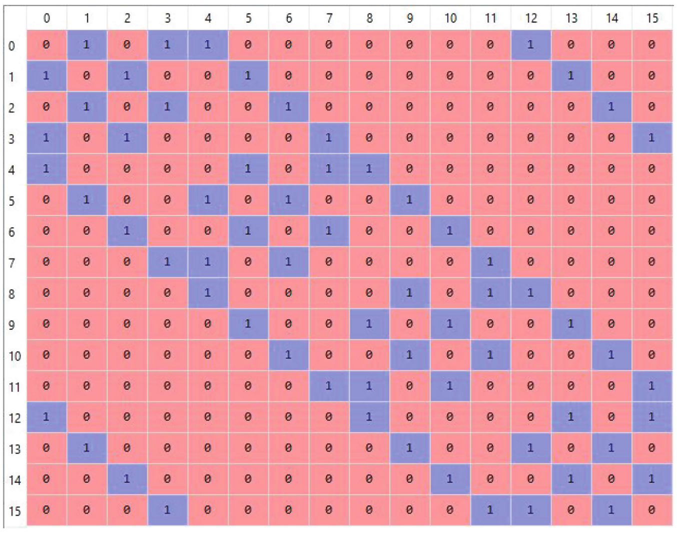
1	0	0	1
0	0	1	0
1	1	1	0
1	0	0	0

(a)



0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

(b)



	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	0	1	1	0	0	0	0	0	0	0	1	0	0	0
1	1	0	1	0	0	1	0	0	0	0	0	0	0	1	0	0
2	0	1	0	1	0	0	1	0	0	0	0	0	0	0	1	0
3	1	0	1	0	0	0	0	1	0	0	0	0	0	0	0	1
4	1	0	0	0	0	1	0	1	1	0	0	0	0	0	0	0
5	0	1	0	0	1	0	1	0	0	1	0	0	0	0	0	0
6	0	0	1	0	0	1	0	1	0	0	1	0	0	0	0	0
7	0	0	0	1	1	0	1	0	0	0	0	1	0	0	0	0
8	0	0	0	0	1	0	0	0	0	1	0	1	1	0	0	0
9	0	0	0	0	0	1	0	0	1	0	1	0	0	1	0	0
10	0	0	0	0	0	0	1	0	0	1	0	1	0	0	1	0
11	0	0	0	0	0	0	0	1	1	0	1	0	0	0	0	1
12	1	0	0	0	0	0	0	0	1	0	0	0	0	1	0	1
13	0	1	0	0	0	0	0	0	0	1	0	0	1	0	1	0
14	0	0	1	0	0	0	0	0	0	0	1	0	0	1	0	1
15	0	0	0	1	0	0	0	0	0	0	0	1	1	0	1	0

(c)

Fig. 2. 2D cellular automaton with 16 cells (4 × 4) and wrapped grid. (a) Example of the grid of cells with states. (b) Indices of the cells and von Neumann counting neighborhood of 2D CA where thick border means the current cell and thin border means the neighbors. (c) Generated adjacency matrix for the described 2D CA.

4 On-Going and Future Applications with EvoDynamic

The method of implementing a CA as an artificial neural network is beneficial for the further development of EvoDynamic framework. Since the implementation of all sparsely connected networks in Table 1 is already planned in forthcoming releases of the Python framework, EvoDynamic shall have a general representation to all of them. Therefore, CAs are treated as ANNs and then can be extended to random Boolean network by shuffling the connections, and to the models that are already ANNs, such as echo state networks and liquid state machines. Moreover, EvoDynamic framework will evolve the connectivity, update and learning rules of the dynamical systems for reservoir computing improvement and physical substrate modeling. This common representation facilitates the evolution of such systems and models which will be guided by several methods that measure the quality of a reservoir or the similarity to a dataset. The following subsections explain two on-going applications with CA that use the EvoDynamic framework.

4.1 State Trajectory

An example of methods to guide the evolution of dynamical system is the state trajectory. This method can be used to cluster similar states for model abstraction and to measure the quality of the reservoir. Therefore, a graph can be formed and analysis can be made by searching for attractors and cycles. For visualization of the state trajectory, we use principal component analysis (PCA) to reduce the dimensionality of the states and present them as a state transition diagram as shown in Fig. 3. The depicted dynamical system is Conway's Game of Life with 7×7 cells and wrapped boundaries. A glider is its initial state (Fig. 3a) and this system cycles over 28 unique states as illustrated in the state transition diagram of Fig. 3l.

4.2 Towards the Evolution for Criticality

Evolution of dynamical systems is a feature currently under development of EvoDynamic framework. The first on-going evolution task of our framework is to find systems with criticality [7] using genetic algorithm, in order to allow for better computational capacity [17]. The first dynamical system for this task is a modified version of stochastic elementary cellular automata (SECA) introduced by Baetens et al. [6]. Our stochastic elementary cellular automaton works as a 1D three neighbors elementary CA, but the next state in time $t+1$ of the central cell c_i is defined by a probability p to be 1 and a probability $1-p$ to be 0 for each of the eight different neighborhood patterns this CA has. Formally, probability p is represented by

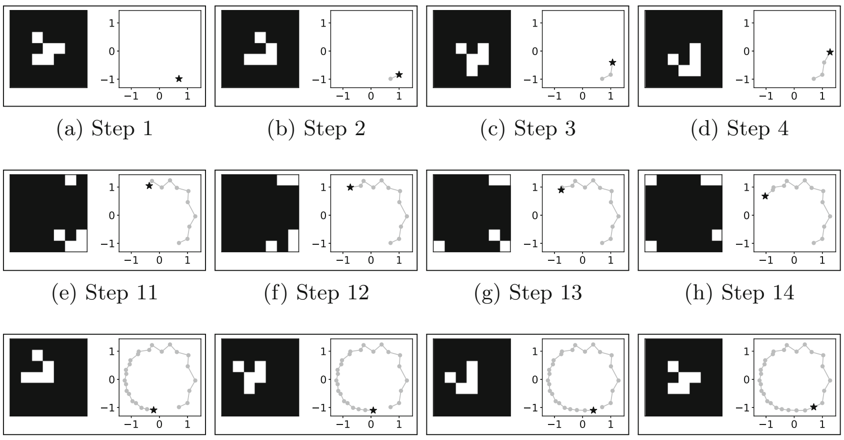
$$p = P(c_{i,t+1} = 1 | N(c_{i,t})) \tag{3}$$

where the neighborhood pattern $N(c_{i,t})$ is denoted as

$$N(c_{i,t}) = (c_{i-1,t}, c_{i,t}, c_{i+1,t}). \tag{4}$$

The genetic algorithm for criticality is guided by a fitness function which mainly verifies if the probability distributions of avalanche size (i.e., cluster size[2]

[2] Cluster size stands for the number of repetitions of a state that happened consecutively without any interruption of another state.



(a) Step 1 (b) Step 2 (c) Step 3 (d) Step 4

(e) Step 11 (f) Step 12 (g) Step 13 (h) Step 14

(i) Step 26 (j) Step 27 (k) Step 28 (l) Step 29

Fig. 3. States of Conway's Game of Life in a 7×7 wrapped lattice alongside their PCA-transformed state transition diagrams of the two first principal components. (a) Initial state is a glider. (a)–(d) Four first steps in this CA. (e)–(h) Four intermediate steps in this CA while reaching the wrapped border. (i)–(l) Four last steps in this CA before repeating the initial state and closing a cycle.

in space) and duration (i.e., cluster size in time) follow a power-law distribution. Such verification can be done by checking how linear is the probability distribution in a log-log plot, by performing goodness-of-fit tests based on the Kolmogorov-Smirnov (KS) statistic and by comparing the power-law model with the exponential model using log-likelihood ratio [9]. For our fitness function, we estimate the candidate distributions with the linear fitting of the first 10 points of the log-log plot using least squares regression, which was verified to be not biased and gives a fast and acceptable estimation of the slope of the power-law distribution [12]. After the linear 10-points fitting, the model is tested using KS statistic. One benefit of using such estimation method is that when the model is not a power-law, the KS statistic reports a large error, i.e., an error greater than one. Another objective in the fitness function is the coefficient of determination [31], but for a complete linear fit of the log-log plot. The fitness function also considers the number of unique states of the stochastic elementary CA, the number of bins in the raw histogram and the value of the estimated power-law exponent. All these fitness function objectives are calculated using a randomly initialized CA of 1,000 cells with wrapped boundaries during 1,000 time-steps. The avalanche size and duration are computed for the cell values 0 and 1, thus producing four different distributions (see Fig. 4) for extracting vectors of their normalized number of histogram bins[3] bin; coefficient of determination R^2 of complete linear fitting; KS statistic D and estimated power-law exponent $\hat{\alpha}$ from the 10-points linear estimation. The fitness score s for each objective is then calculated by the following equations:

$$bin_s = \tanh(5 * (0.9 * \max(bin) + 0.1 * \mathrm{mean}(bin))), \tag{5}$$

$$R^2_s = \mathrm{mean}(R^2), \tag{6}$$

$$D_s = \exp(-(0.9 * \min(D) + 0.1 * \mathrm{mean}(D))), \tag{7}$$

$$\hat{\alpha}_s = \mathrm{mean}(\hat{\alpha}), \tag{8}$$

$$unique_s = \frac{\#uniqueStates}{\#timesteps}. \tag{9}$$

The (5)–(9) are all objective values for calculating the fitness score s. Those values are real numbers between zero and one, except the score for the estimated power-law exponent $\hat{\alpha}_s$, and they have weights attributed to them regarding their level of importance and for compensating small and large values. The following equation denotes how the fitness score s is calculated:

$$s = 10 * bin_s + 10 * R^2_s + 10 * D_s + 0.1 * \hat{\alpha}_s + 10 * unique_s. \tag{10}$$

The genetic algorithm has 40 individuals that evolve through 100 generations. The optimization performed by GA is to maximize the fitness score. The genome of the individuals has eight real number genes with a value range between zero and one. Each gene represents the probability of the next state becoming one

Table 2. Best individual

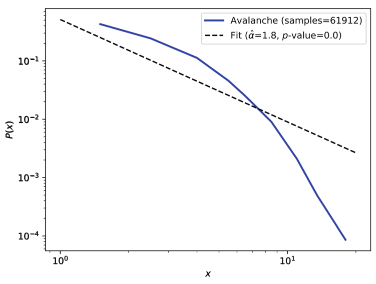
Neighborhood $N(c_{i,t})$	Probability p
(0,0,0)	0.103009
(0,0,1)	0.536786
(0,1,0)	0.216794
(0,1,1)	0.393468
(1,0,0)	0.679836
(1,0,1)	0.175458
(1,1,0)	0.724778
(1,1,1)	1.000000

[3] The actual number of histogram bins is normalized or divided by the possible total number of bins.

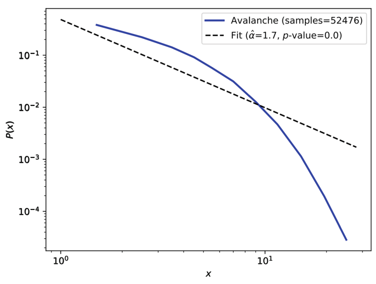
Table 3. Fitness score of the best individual

Objective	Score
$10 * bin_s$	9.780749590096136
$10 * R_s^2$	8.832520186440096
$10 * D_s$	9.655719560019996
$0.1 * \hat{\alpha}_s$	0.18022617747972156
$10 * unique_s$	10.0
s	**38.44921551403595**

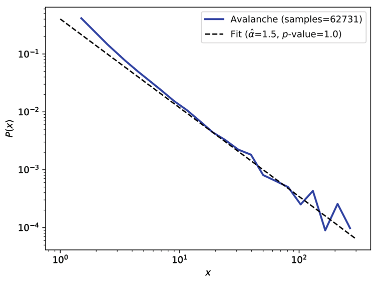
(i.e., p in (3)) for its respective neighborhood pattern. The selection of two parents is done by deterministic tournament selection [11]. After that, the crossover between the genomes of the parents can happen with probability 0.8, then each gene can be exchanged with probability 0.5. Afterward, a mutation occurs to a gene with probability 0.1. This mutation adds a random value from a normal distribution with mean and standard deviation equals to, respectively, 0 and



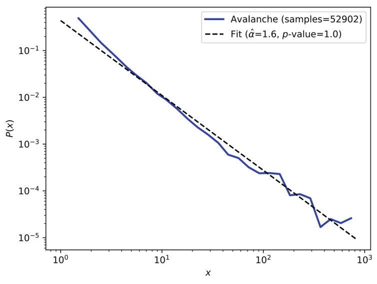
(a) Avalanche size of state 0



(b) Avalanche duration of state 0



(c) Avalanche size of state 1



(d) Avalanche duration of state 1

Fig. 4. Avalanche size and duration of the two states 0 and 1 of the evolved stochastic elementary CA.

0.2. The mating process of the two parents produces an offspring of two new individuals who replace the parents in the next generation.

An example of an evolved genome for the best resulting individual is presented in Table 2. The fitness score s and all objective scores with their respective weights for calculating s are in Table 3.

With the genome or probabilities of the eight different neighborhood patterns of the best evolved individual, we can produce the log-log plots of the probability distribution of avalanche size and duration for the states zero and one. Such plots are depicted in Fig. 4. The p-value of goodness-of-fit test is calculated using 1,000 randomly generated data with 10,000 samples applying the power-law exponent $\hat{\alpha}$ estimated by maximum likelihood estimation method with minimum x of the distribution fixed to 1. The Figs. 4a and b show the avalanche size and

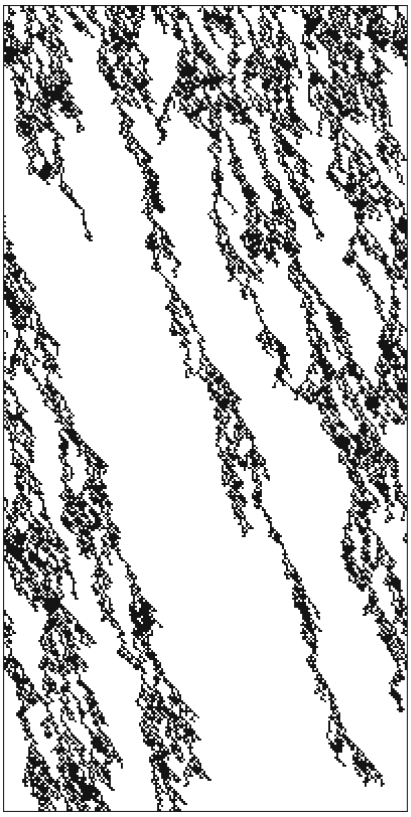


Fig. 5. Sample of the best evolved stochastic elementary CA of 200 cells (horizontal axis) randomly initialized with wrapped boundaries through 400 time-steps (vertical axis).

duration for the state 0 or black. They present distributions that are not a power-law because they do not fit the power-law estimation (the black dashed line). Moreover, the p-value is equal to 0.0 which proves that those two distributions are not a power-law. The Figs. 4c and d present the avalanche size and duration for the state 1 or white. Those distributions follow a power-law because, visually, the estimated power-law distribution fits the empirical probability distribution and, quantitatively, the p-value is equal to 1.0 which means that 100% of the KS statistic of the generated data is greater than the KS statistic of the empirical distribution of avalanche size and duration of state 1. The number of samples in those distributions (62,731 for avalanche size and 52,902 for avalanche duration) confirms that the p-value is trustworthy. Such power-law analysis is performed by utilizing the powerlaw Python library [5]. It is important to warn that high fitness scores do not mean p-values closer to 1.0 and the goodness-of-fit test is not part of the fitness score because it is a slow process.

A sample of the resulting stochastic elementary cellular automaton of the best individual is illustrated in Fig. 5. This CA, as seen, has no static nor periodic states, and no random evolution of its states. Therefore, this dynamical system is between a strongly and weakly coupled substrate. Therefore, the CA presents patterns or structures that mean the cells are interdependent in this system.

5 Conclusion

In this paper, we present an alternative method to implement a cellular automaton. This allows any CA to be computed as an artificial neural network. That means, any lookup table can be an activation function, and any neighborhood and dimensionality can be represented as a weight matrix. Therefore, this will help to extend the CA implementation to more complex dynamical systems, such as random Boolean networks, echo state networks and liquid state machines. Furthermore, the EvoDynamic framework is built on a deep learning library, TensorFlow, which permits the acceleration and parallelization of matrix operations when applied on computational platforms with fast CPUs and GPUs. The planned future implementations of EvoDynamic are presented and discussed. The state trajectory is an important feature for the targeted future tasks. The evolution with genetic algorithm towards criticality of stochastic CA is showing promising results and our next goal can be for self-organized criticality. The future work for the CA implementation is to develop algorithms to procedurally generate weighted adjacency matrices for 3D and multidimensional cellular automata with different types of cells, such as the cells with hexagonal shape in a 2D CA.

Acknowledgments. We thank Kristine Heiney for thoughtful discussions about self-organized criticality.

References

1. SOCRATES – Self-Organizing Computational substRATES. https://www.ntnu.edu/socrates
2. Conway's game of life implemented using tensorflow 2d convolution function (2016). https://github.com/conceptacid/conv2d_life
3. Aaser, P., et al.: Towards making a cyborg: a closed-loop reservoir-neuro system. In: The 2018 Conference on Artificial Life: A Hybrid of the European Conference on Artificial Life (ECAL) and the International Conference on the Synthesis and Simulation of Living Systems (ALIFE), no. 29, pp. 430–437 (2017). https://doi.org/10.1162/isal_a_072
4. Abadi, M., et al.: Tensorflow: a system for large-scale machine learning. In: 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16), pp. 265–283. USENIX Association, Savannah, GA (2016)
5. Alstott, J., Bullmore, E., Plenz, D.: Powerlaw: a python package for analysis of heavy-tailed distributions. PLOS ONE **9**(1), 1–11 (2014). https://doi.org/10.1371/journal.pone.0085777
6. Baetens, J.M., Van der Meeren, W., De Baets, B.: On the dynamics of stochastic elementary cellular automata. J. Cell. Automata **12**, 63–80 (2016)
7. Bak, P., Tang, C., Wiesenfeld, K.: Self-organized criticality: an explanation of the 1/f noise. Phys. Rev. Lett. **59**, 381–384 (1987). https://doi.org/10.1103/PhysRevLett.59.381
8. Broersma, H., Miller, J.F., Nichele, S.: Computational Matter: Evolving Computational Functions in Nanoscale Materials. In: Adamatzky, A. (ed.) Advances in Unconventional Computing. ECC, vol. 23, pp. 397–428. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-33921-4_16
9. Clauset, A., Shalizi, C.R., Newman, M.E.: Power-law distributions in empirical data. SIAM Rev. **51**(4), 661–703 (2009)
10. Gershenson, C.: Introduction to random boolean networks (2004). arXiv preprint nlin/0408006
11. Goldberg, D.E., Deb, K.: A comparative analysis of selection schemes used in genetic algorithms. In: Foundations of Genetic Algorithms, vol. 1, pp. 69–93. Elsevier (1991)
12. Goldstein, M.L., Morris, S.A., Yen, G.G.: Problems with fitting to the power-law distribution. Eur. Phys. J. B-Condens. Matter Complex Syst. **41**(2), 255–258 (2004)
13. Jaeger, H., Haas, H.: Harnessing nonlinearity: predicting chaotic systems and saving energy in wireless communication. Science **304**(5667), 78–80 (2004). https://doi.org/10.1126/science.1091277
14. Jensen, J.H., Folven, E., Tufte, G.: Computation in artificial spin ice. In: The 2018 Conference on Artificial Life: A Hybrid of the European Conference on Artificial Life (ECAL) and the International Conference on the Synthesis and Simulation of Living Systems (ALIFE), no. 30, pp. 15–22 (2018). https://doi.org/10.1162/isal_a_00011
15. Kaneko, K.: Overview of coupled map lattices. Chaos: Interdisc. J. Nonlinear Sci. **2**(3), 279–282 (1992)
16. Konkoli, Z., Nichele, S., Dale, M., Stepney, S.: Reservoir Computing with Computational Matter. Comput. Matter. NCS, pp. 269–293. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-65826-1_14

17. Langton, C.G.: Computation at the edge of chaos: phase transitions and emergent computation. Phys. D: Nonlinear Phenom. **42**(1), 12–37 (1990). https://doi.org/10.1016/0167-2789(90)90064-V
18. Maass, W., Markram, H.: On the computational power of circuits of spiking neurons. J. Comput. Syst. Sci. **69**(4), 593–616 (2004). https://doi.org/10.1016/j.jcss.2004.04.001
19. Nichele, S., Tufte, G.: Trajectories and attractors as specification for the evolution of behaviour in cellular automata. In: IEEE Congress on Evolutionary Computation, pp. 1–8, July 2010. https://doi.org/10.1109/CEC.2010.5586115
20. Nichele, S., Farstad, S.S., Tufte, G.: Universality of evolved cellular automata in-materio. Int. J. Unconv. Comput. **13**(1) (2017)
21. Nichele, S., Gundersen, M.S.: Reservoir computing using nonuniform binary cellular automata. Complex Syst. **26**(3), 225–245 (2017). https://doi.org/10.25088/complexsystems.26.3.225
22. Nichele, S., Molund, A.: Deep learning with cellular automaton-based reservoir computing. Complex Syst. **26**(4), 319–339 (2017). https://doi.org/10.25088/complexsystems.26.4.319
23. Nichele, S., Tufte, G.: Genome parameters as information to forecast emergent developmental behaviors. In: Durand-Lose, J., Jonoska, N. (eds.) UCNC 2012. LNCS, vol. 7445, pp. 186–197. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-32894-7_18
24. Rendell, P.: Turing Universality of the Game of Life, pp. 513–539. Springer, London (2002). https://doi.org/10.1007/978-1-4471-0129-1_18
25. Schrauwen, B., Verstraeten, D., Van Campenhout, J.: An overview of reservoir computing: theory, applications and implementations. In: Proceedings of the 15th European Symposium on Artificial Neural Networks 2007, pp. 471–482 (2007)
26. Subramoney, A., Scherr, F., Maass, W.: Reservoirs learn to learn (2019). arXiv preprint arXiv:1909.07486
27. Tanaka, G., et al.: Recent advances in physical reservoir computing: a review. Neural Netw. **115**, 100–123 (2019). https://doi.org/10.1016/j.neunet.2019.03.005
28. TensorFlow: tf.sparse.sparse_dense_matmul — tensorflow core r1.14 — tensorflow. https://www.tensorflow.org/api_docs/python/tf/sparse/sparse_dense_matmul
29. Toffoli, T., Margolus, N.: Cellular Automata Machines: A New Environment for Modeling. MIT press, Cambridge (1987)
30. Wolfram, S.: A New Kind of Science, vol. 5. Wolfram Media, Champaign (2002)
31. Wright, S.: Correlation and causation. J. Agric. Res. **20**, 557–580 (1921)

Paper A.2

A neuro-inspired general framework for the evolution of stochastic dynamical systems: cellular automata, random Boolean networks and echo state networks towards criticality

(Pontes-Filho et al., 2020)

Author(s):

Sidney Pontes-Filho, Pedro Lind, Anis Yazidi, Jianhua Zhang, Hugo Hammer, Gustavo Mello, Ioanna Sandvig, Gunnar Tufte and Stefano Nichele

Published in journal:

Cognitive Neurodynamics, vol. 14, pp. 657–674, 2020

Paper A.2
Pontes-Filho et al. (2020)

Cognitive Neurodynamics
https://doi.org/10.1007/s11571-020-09600-x

RESEARCH ARTICLE

A neuro-inspired general framework for the evolution of stochastic dynamical systems: Cellular automata, random Boolean networks and echo state networks towards criticality

Sidney Pontes-Filho[1,2] · Pedro Lind[1] · Anis Yazidi[1] · Jianhua Zhang[1] · Hugo Hammer[1] · Gustavo B. M. Mello[1] · Ioanna Sandvig[3] · Gunnar Tufte[2] · Stefano Nichele[1,4]

Received: 30 December 2019 / Revised: 8 May 2020 / Accepted: 14 May 2020

Abstract
Although deep learning has recently increased in popularity, it suffers from various problems including high computational complexity, energy greedy computation, and lack of scalability, to mention a few. In this paper, we investigate an alternative brain-inspired method for data analysis that circumvents the deep learning drawbacks by taking the actual dynamical behavior of biological neural networks into account. For this purpose, we develop a general framework for dynamical systems that can evolve and model a variety of substrates that possess computational capacity. Therefore, dynamical systems can be exploited in the reservoir computing paradigm, i.e., an untrained recurrent nonlinear network with a trained linear readout layer. Moreover, our general framework, called EvoDynamic, is based on an optimized deep neural network library. Hence, generalization and performance can be balanced. The EvoDynamic framework contains three kinds of dynamical systems already implemented, namely cellular automata, random Boolean networks, and echo state networks. The evolution of such systems towards a dynamical behavior, called criticality, is investigated because systems with such behavior may be better suited to do useful computation. The implemented dynamical systems are stochastic and their evolution with genetic algorithm mutates their update rules or network initialization. The obtained results are promising and demonstrate that criticality is achieved. In addition to the presented results, our framework can also be utilized to evolve the dynamical systems connectivity, update and learning rules to improve the quality of the reservoir used for solving computational tasks and physical substrate modeling.

Keywords Dynamical systems · Implementation · Reservoir computing · Evolution · Criticality

Introduction

Every day, humans produce exabytes of data and this trend is growing due to emerging technologies, such as 5G and the Internet of Things (McAfee et al. 2012). Given that the main computing technology is based on von Neumann architecture, the analysis of enormous amounts of data is challenging even for the popular deep learning methods (Oussous et al. 2018). Deep learning is a powerful data analysis tool, but it has some problems, including high energy consumption, and lack of scalability and flexibility. Therefore, a new type of architecture may be required to alleviate such problems, in particular energy efficiency, scalability, adaptability, and robustness. The brain, or rather, an architecture inspired by the brain, can be this new architecture. This computing organ is energy efficient,

✉ Sidney Pontes-Filho
sidneyp@oslomet.no

[1] Department of Computer Science, Oslo Metropolitan University, Oslo, Norway

[2] Department of Computer Science, Norwegian University of Science and Technology, Trondheim, Norway

[3] Department of Neuromedicine and Movement Science, Norwegian University of Science and Technology, Trondheim, Norway

[4] Department of Holistic Systems, Simula Metropolitan, Oslo, Norway

Published online: 11 June 2020

adaptable, robust, and can perform parallel processing through local interactions (Markram et al. 2011).

Artificial systems with similar dynamical properties to the brain exist, such as cellular automata (Wolfram 2002), random Boolean networks (Gershenson 2004), and artificial neural networks (Jaeger and Haas 2004; Maass and Markram 2004). However, their dynamics are difficult to program or control in order to perform useful computation. In such systems, Langton (1990) suggests that computational properties are connected to the "edge of chaos" behavior, a range of dynamical behaviors between order and disorder. In other words, they are systems critically near a phase transition. If the attractors of the system are in the critical state, this characteristic is called self-organized criticality (Bak et al. 1987). Systems with self-organized criticality have a common feature, i.e., power-law correlations in time or space that extend over several scales. Moreover, biological neural networks have been shown to self-organize into criticality, which is evaluated by the power-law distribution of neuronal avalanches (Heiney et al. 2019; Tetzlaff et al. 2010; Yada et al. 2017). Another important aspect of the computation performed in a dynamical system is the trajectory of system states traversed during the computation (Nichele and Tufte 2010). Such a trajectory may be guided by system parameters (Nichele and Tufte 2012).

Table 1 presents some computing systems that are capable of giving rise to the emergence of complex dynamics. The approaches in such a table (and the work presented herein) are extensions to previous works (Pontes-Filho et al. 2019a, b). Dynamical systems with complex behavior can be availed by reservoir computing, which is a paradigm that resorts to dynamical systems to simplify complex nonlinear data. Such simplification means that reservoir computing utilizes the nonlinear dynamical system to perform a nonlinear transformation from nonlinear data to higher dimensional linear data. Such linearized data can be applied in linear machine learning methods which are faster for training and computing because they have less trainable variables and operations. Hence, reservoir computing is more energy efficient than deep learning methods and it can even yield competitive results, especially for temporal data (Schrauwen et al. 2007; Tanaka et al. 2019). Basically, reservoir computing exploits a dynamical system that possesses the echo state property and fading memory, where the internals of the reservoir are untrained and the training only happens at the linear readout stage (Konkoli et al. 2018).

Reservoir computers are most useful when their substrates' dynamics are at the "edge of chaos" (Langton 1990). A simple computing system used as a reservoir is a cellular automaton (CA) (Nichele and Gundersen 2017; Nichele and Molund 2017). A CA consists of a grid of cells with a finite number of states that change according to simple rules depending on the neighborhood and own state in discrete time-steps. Other systems can also exhibit similar dynamics. The coupled map lattice (Kaneko 1992) is very similar to CA, the only exception is that the coupled map lattice has continuous states which are updated by a recurrence equation involving the neighborhood. A random Boolean network (RBN) (Gershenson 2004) is a generalization of CA where random connectivity exists. An echo state network (ESN) (Jaeger and Haas 2004) is an artificial neural network (ANN) with random topology. A spiking cellular automaton (Bailey 2010) is a CA whose cells are spiking neurons that communicate through discrete-events (spikes) over continuous time. A spiking neuron is a model of the biological neuron found in the brain. A lattice of ordinary differential equations (Chow et al. 1996; Larter et al. 1999) is a cellular automaton where state and time are continuous and updated by ordinary differential equations (ODEs). A liquid state machine (Maass and Markram 2004) is an echo state network with spiking neurons. ODEs in complex topology are similar to the lattice differential equations, but the connectivity is random. Moreover, computation in dynamical systems may be carried out in physical substrates (Tanaka et al. 2019), such as in-vitro networks of biological neurons (Aaser et al. 2017) or in nanoscale materials (Broersma et al. 2017). Finding the correct abstraction for the computation in a dynamical system, e.g. CA, is still an open research problem (Nichele et al. 2017).

Table 1 Examples of dynamical systems

Dynamical system	State	Time	Connectivity
Cellular automaton	Discrete	Discrete	Regular
Coupled map lattice	Continuous	Discrete	Regular
Random Boolean network	Discrete	Discrete	Random
Echo state network	Continuous	Discrete	Random
Spiking cellular automaton	Discrete	Continuous	Regular
Lattice differential equations	Continuous	Continuous	Regular
Liquid state machine	Discrete	Continuous	Random
ODEs in complex topology	Continuous	Continuous	Random

One of our goals is to simulate all of these computing systems in a single general framework. Since generalization affects performance, we counterbalance it by using an optimized parallel library, such as the TensorFlow deep neural network framework (Abadi et al. 2016). To be able to exploit this library, a dynamical system is represented by a weighted adjacency matrix, such as a graph, and calculated as an artificial neural network, then taking advantage of the library's optimization. Moreover, the weighted adjacency matrix of a dynamical system with complex dynamics is normally sparse. Thus, the choice of TensorFlow is advantageous because of its optimized methods and data types for sparse matrices or tensors. Another goal is to tune dynamical systems to reach the critical point at the "edge of chaos", criticality, or even to search for systems with self-organized criticality. Systems in self-organized criticality may be better suited for performing useful computation in reservoir computing. To accomplish our goals, the presented general framework for dynamical systems, called EvoDynamic[1], aims at evolving (i.e., using evolutionary algorithms) the connectivity, update and learning rules of sparsely connected networks to improve their usage for reservoir computing guided by the echo state property, fading memory, state trajectory, and other quality measurements. Such improvement of reservoirs is similarly applied in (Subramoney et al. 2019), where the internal connectivity of a reservoir is trained to increase its performance to several tasks. To verify that, we evolved three different stochastic dynamical systems, namely a cellular automaton, random Boolean network, and echo state network, towards criticality using a genetic algorithm. In the previous works (Pontes-Filho et al. 2019a, b), only cellular automaton is investigated and the fitness function for the genetic algorithm in (Pontes-Filho et al. (2019a) is less effective than the one proposed in this work. The evolution of these three stochastic dynamical systems was guided by fitting a power-law model into the distributions of avalanche size and duration. Moreover, for future work, evolution will model the dynamics and behavior of physical reservoirs, such as in-vitro biological neural networks interfaced with microelectrode arrays, and nanomagnetic ensembles. These two substrates have real applicability as reservoirs. For example, the former substrate is applied to control a robot, in effect making it a cyborg, a closed-loop biological-artificial neuro-system (Aaser et al. 2017), and the latter possesses computation capability as shown by a square lattice of nanomagnets (Jensen et al. 2018). These substrates are the main interest of the SOCRATES project (https://www.ntnu.edu/socrates) which aims to explore a dynamic, robust, and energy efficient hardware for data analysis.

[1] EvoDynamic open-source repository on https://github.com/SocratesNFR/EvoDynamic.

This paper is organized as follows. Section 2 describes our method of computing dynamical systems in a generalized manner and the approach of evolving three stochastic dynamical systems towards criticality. Section 3 presents the results obtained from the methods. Section 4 discusses the experimental results. Section 5 states the initial advances and future plan for the EvoDynamic framework and Sect. 6 concludes this paper.

Methods

There are two main methods described in this section. One method is to simulate dynamical systems in a general manner, which is very similar to simulating an artificial neural network, and no training is needed. The other method is to evolve three stochastic dynamical systems towards criticality. The three systems are based on cellular automata, random Boolean networks, and echo state networks, respectively.

General framework for dynamical systems

Generalization is necessary to be able to simulate several dynamical systems with a single implementation. Therefore, our idea is to procedurally modify the computation of an artificial neural network to fit the dynamics of the desired dynamical system. In order to do that, modifications are introduced in the weighted adjacency matrix $\mathbf{A}$ and the mapping function f. $\mathbf{A}$ and f are analogous, respectively, to the weight matrix and activation function of artificial neural networks. The weighted adjacency matrix $\mathbf{A}$ and the mapping function f are used to compute the next state in time $t+1$ from the current state in time t of the components of the dynamical system that are called cells $\mathbf{c}$. The equation for that is

$$\mathbf{c}_{t+1} = f(\mathbf{A} \cdot \mathbf{c}_t). \qquad (1)$$

This is similar to the equation of the forward pass of an artificial neural network but without the bias. The next states of the cells $\mathbf{c}_{t+1}$ are calculated from the result of the mapping function f which receives as argument the dot product between the weighted adjacency matrix $\mathbf{A}$ and the current states of the cells $\mathbf{c}_t$. The vector $\mathbf{c}$ is always a column vector of size $len(\mathbf{c}) \times 1$, and $\mathbf{A}$ is a matrix of size $len(\mathbf{c}) \times len(\mathbf{c})$. Hence the result of $\mathbf{A} \cdot \mathbf{c}$ is also a column vector of size $len(\mathbf{c}) \times 1$ as $\mathbf{c}$.

Dynamical systems that possess a critical regime are often sparsely connected networks. Since the EvoDynamic framework is implemented on TensorFlow, the data type of the weighted adjacency matrix $\mathbf{A}$ is preferably a `SparseTensor`. A dot product with such a data type can result in up to 9 times faster execution than the dense counterpart. However, this depends on the configuration of the tensors (or, in our case, the adjacency matrices) (https://www.tensorflow.org/api_docs/python/tf/sparse/sparse_dense_matmul).

The details of how this general framework is used for the three stochastic dynamical systems that are evolved towards criticality are described in the following sections.

Cellular automata in the general framework

The implementation of a cellular automaton in our general framework requires the procedural generation of the weighted adjacency matrix of its grid. In this way, any lattice type or multidimensional CAs can be implemented using our framework. Algorithm 1 generates the weighted adjacency matrix for one-dimensional CA, such as the elementary cellular automaton (Wolfram 2002), where *widthCA* is the width or number of cells of a unidimensional CA and the vector **neighborhood** describes the region around the center cell. The connection weights depend on the type of update rule as previously explained. For example, in the case of an elementary CA, **neighborhood** $= [4\ 2\ 1]$ (acquired from (2)). *indexNeighborCenter* is the index of the center cell in the **neighborhood** whose starting index is zero. *isWrappedGrid* is a Boolean value that works as a flag for adding a wrapped grid or not. A wrapped grid for one-dimensional CA means that the initial and final cells are neighbors. With all these parameters, Algorithm 1 creates an adjacency matrix by looping over the indices of the cells (from zero to *numberOfCells* $-$ 1) with an inner loop for the indices of the neighbors. If the selected *currentNeighbor* is a non-zero value and its indices do not affect the boundary condition, then the value of *currentNeighbor* is assigned to the adjacency matrix **A** in the indices that correspond to the connection between the current cell in the outer loop and the actual index of *currentNeighbor*. Finally, this procedure returns the adjacency matrix **A**.

Algorithm 1 Generation of weighted adjacency matrix for 1D cellular automaton

```
1: procedure GENERATECA1D
2:     $numberOfCells \leftarrow widthCA$
3:     $\mathbf{A} \leftarrow \mathbf{0}^{numberOfCells \times numberOfCells}$          ▷ Adjacency matrix initialization
4:     for $i \leftarrow \{0..(numberOfCells - 1)\}$ do
5:         for $j \leftarrow \{-indexNeighborCenter..(len(neighborhood) - indexNeighborCenter - 1)\}$ do
6:             $currentNeighbor \leftarrow neighborhood_{j+indexNeighborCenter}$
7:             if $currentNeighbor \neq 0 \wedge (isWrappedGrid \vee (\neg isWrappedGrid \wedge (0 \leq (i+j) < widthCA))$ then
8:                 $\mathbf{A}_{i,((i+j) \bmod widthCA)} \leftarrow currentNeighbor$
9:     return A
```

Minor adjustments need to be made to the algorithm to procedurally generate an adjacency matrix for 2D CA instead of 1D CA. Algorithm 2 shows the procedure for two-dimensional CA, such as Conway's Game of Life. In this case, the height of the CA is an argument passed as *heightCA*. **Neighborhood** is a 2D matrix and **indexNeighborCenter** is a vector of two components meaning the indices of the center of **Neighborhood**. This procedure is similar to the one in Algorithm 1, but it contains one more loop for the additional dimension.

Algorithm 2 Generation of adjacency matrix of 2D cellular automaton

```
1: procedure GENERATECA2D
2:     $numberOfCells \leftarrow widthCA * heightCA$
3:     $\mathbf{A} \leftarrow \mathbf{0}^{numberOfCells \times numberOfCells}$        ▷ Adjacency matrix initialization
4:     $widthNB, heightNB \leftarrow shape(\mathbf{Neighborhood})$
5:     for $i \leftarrow \{0..(numberOfCells - 1)\}$ do
6:         for $j \leftarrow \{-\mathbf{indexNeighborCenter}_0..(widthNB - \mathbf{indexNeighborCenter}_0 - 1)\}$ do
7:             for $k \leftarrow \{-\mathbf{indexNeighborCenter}_1..(heightNB - \mathbf{indexNeighborCenter}_1 - 1)\}$ do
8:                 $currentNeighbor \leftarrow Neighborhood_{j+indexNeighborCenter}$
9:                 if $currentNeighbor \neq 0 \wedge (isWrappedGrid \vee (\neg isWrappedGrid \wedge (0 \leq ((i \bmod heightCA) + j) < widthCA) \wedge (0 \leq (\lfloor i/widthCA \rfloor + k) < heightCA))$ then
10:                    $\mathbf{A}_{i,(((i+k) \bmod widthCA)+((\lfloor i/widthCA \rfloor + j) \bmod heightCA)*widthCA)} \leftarrow currentNeighbor$
11:    return $\mathbf{A}$
```

The update rule of the CA alters the weights of the connections in the adjacency matrix. For example, Conway's Game of Life (Rendell 2002) is a CA whose cells have two states meaning "dead" (zero) or "alive" (one), and the update rule is based on the number of "alive" cells in the neighborhood. Therefore, for counting the number of alive "neighbors", the weights in the adjacency matrix are one for connection and zero for no connection, as in an ordinary adjacency matrix. Such a matrix facilitates the description of the update rule for counting the number of "alive" neighbors because the result of the dot product between the adjacency matrix and the cell state vector is the vector that contains the number of "alive" neighbors for each cell. This is shown in Fig. 1 for a 2D CA of 16 cells (4×4), wrapped grids and modification in the original neighborhood (Fig. 1a), cells' indices and von Neumann neighborhood (Fig. 1b), and its weighted adjacency matrix (acquired from Algorithm 2) which is used to compute the number of "alive" neighbors for this CA (Fig. 1c).

Another example where the CA's update rule affects the weighted adjacency matrix is when the pattern of the neighborhood influences the update rule, such as in an elementary cellular automaton (Wolfram 2002). To do that, each cell has its neighbors encoded as a n-ary string where n means the number of states that a cell can have. Hence, the weights of the connections with the neighbors are n-base identifiers and are calculated by

$$neighbor_i = n^i, \forall i \in \{0..len(\mathbf{neighbors}) - 1\} \quad (2)$$

where **neighbors** is a vector of the cell's neighbors. In the adjacency matrix, each neighbor receives a weight according to (2). The result of the dot product with such a weighted adjacency matrix is a vector that consists of unique integers per neighborhood pattern. Thus, the mapping function is a lookup table from integer (i.e., pattern code) to next state. This is depicted in Fig. 2 for a 1D elementary cellular automaton of 16 cells and wrapped grids (Fig. 2a), cells' indices and neighborhood (Fig. 2b), and its weighted adjacency matrix (acquired from Algorithm 1) being used to calculate the values for the mapping function (Fig. 2c).

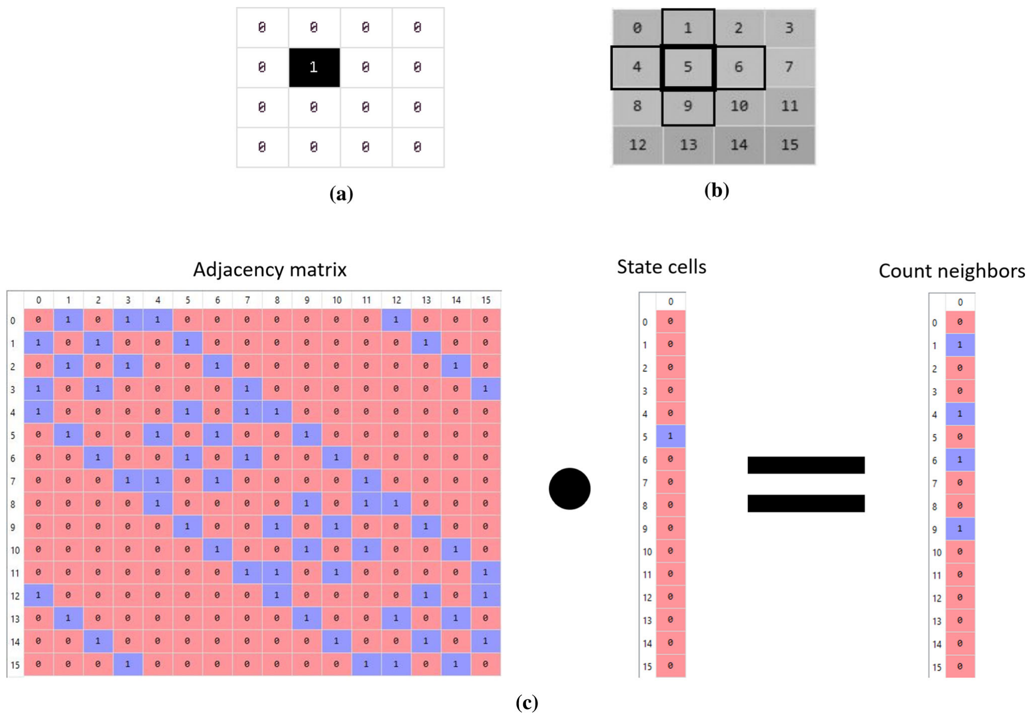
The mapping function for CA is different from the activation function used for ANN. For CA, it contains the update rules that verify the vector returned by the dot product between the weighted adjacency matrix and the vector of states. Normally, the update rules of the CA are implemented as a lookup table from neighborhood to next state. In our implementation, the lookup table maps the resulting vector of the dot product to the next state of the central cell.

Random Boolean networks in the general framework

A random Boolean network (RBN) is an extension of cellular automata (Gershenson 2004) where the regular grid is replaced by random connections between the nodes or cells. An RBN has a similar update function to a CA whose cells consider the states of each of its neighbors, such as the neighborhood pattern of an elementary CA. Basically, a weighted adjacency matrix of a random Boolean network is acquired by shuffling the rows of the matrix for an elementary CA. Figure 3 illustrates the weighted adjacency matrix and the graph of a random Boolean network whose cells are randomly connected to three other cells. The difference between Figs. 2c and 3a shows how the method for elementary CA is adjusted for a random Boolean network.

Echo state networks in the general framework

Our general framework for dynamical systems is based on the computation of artificial neural networks. Since an echo state network (ESN) (Jaeger and Haas 2004) is a type of



Adjacency matrix

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	0	1	1	0	0	0	0	0	0	0	1	0	0	0
1	1	0	1	0	0	1	0	0	0	0	0	0	0	1	0	0
2	0	1	0	1	0	0	1	0	0	0	0	0	0	0	1	0
3	1	0	1	0	0	0	0	1	0	0	0	0	0	0	0	1
4	1	0	0	0	0	1	0	1	1	0	0	0	0	0	0	0
5	0	1	0	0	1	0	1	0	0	1	0	0	0	0	0	0
6	0	0	1	0	0	1	0	1	0	0	1	0	0	0	0	0
7	0	0	0	1	1	0	1	0	0	0	0	1	0	0	0	0
8	0	0	0	0	1	0	0	0	0	1	0	1	1	0	0	0
9	0	0	0	0	0	1	0	0	1	0	1	0	0	1	0	0
10	0	0	0	0	0	0	1	0	0	1	0	1	0	0	1	0
11	0	0	0	0	0	0	0	1	1	0	1	0	0	0	0	1
12	1	0	0	0	0	0	0	0	1	0	0	0	0	1	0	1
13	0	1	0	0	0	0	0	0	0	1	0	0	1	0	1	0
14	0	0	1	0	0	0	0	0	0	0	1	0	0	1	0	1
15	0	0	0	1	0	0	0	0	0	0	0	1	1	0	1	0

State cells

	0
0	0
1	0
2	0
3	0
4	0
5	1
6	0
7	0
8	0
9	0
10	0
11	0
12	0
13	0
14	0
15	0

Count neighbors

	0
0	0
1	1
2	0
3	0
4	1
5	0
6	1
7	0
8	0
9	1
10	0
11	0
12	0
13	0
14	0
15	0

Fig. 1 Example of using matrix multiplication for computing a 2D cellular automaton with 16 cells (4×4) and wrapped grid. **a** Example of the grid of cells with states. State 0 means dead or non-occupied cell and state 1 stands for alive or occupied cell. **b** Indices of the cells and von Neumann counting neighborhood of 2D CA where thick border means the current cell and thin border means the neighbors. **c** Illustration of matrix multiplication between adjacency matrix of the 2D CA and the state vector of the flattened 2D CA, resulting in a vector that contains the number of alive neighbors for each cell. Please note that an alive cell does not count itself as an alive neighbor

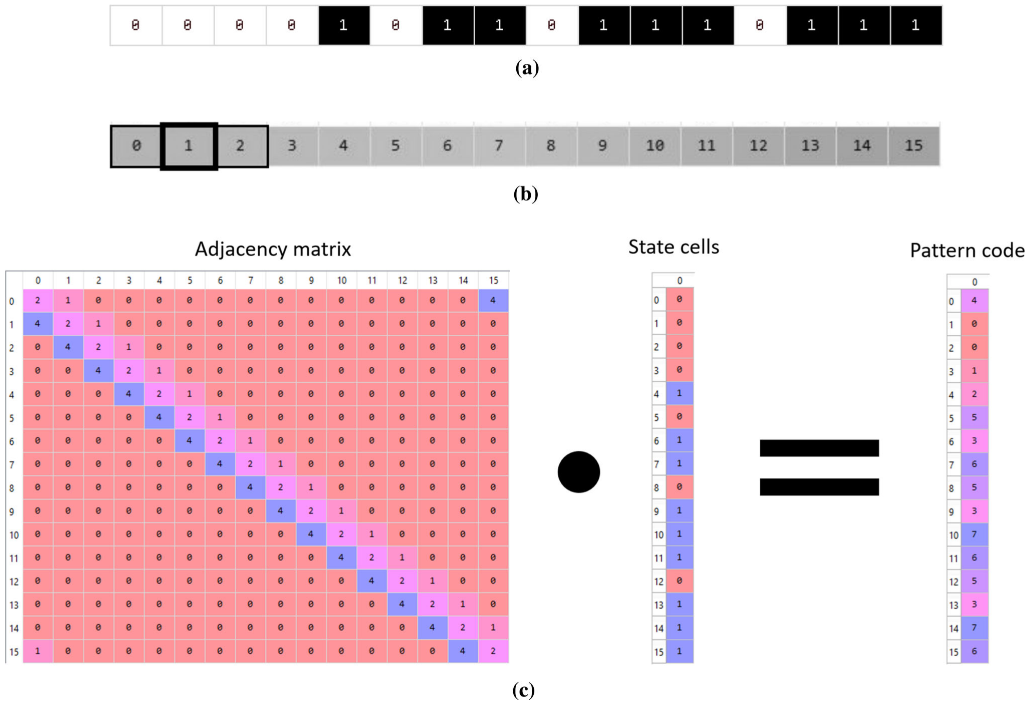
artificial neural network, the weighted adjacency matrix is the usual weight matrix and the mapping function is one of the several activation functions that can be used for the neurons in an artificial neural network, such as sigmoid, hyperbolic tangent and rectified linear unit (LeCun et al. 2015). Note that in an ESN, the connection weights are randomly initialized. This is depicted in Fig. 4 where an echo state network of 10 cells or neurons are randomly connected with a certain sparsity. The color of the cells shows their states between 0 and 1 in grayscale. The edges are colored as red and blue to represent the negative and positive weights, respectively. The thickness of the edges is proportional to the weight value of the connections.

Evolution of stochastic dynamical systems towards criticality

Using the previously explained general framework, we simulate three stochastic dynamical systems, namely cellular automata, random Boolean networks, and echo state networks. The evolution through genetic algorithm aims to find systems with criticality (Bak et al. 1987), in order to improve computational capacity (Langton 1990).

The stochastic dynamical systems

The first stochastic dynamical system towards criticality is a modified version of stochastic elementary cellular automata (SECA) introduced by Baetens et al. (2016). Our stochastic elementary cellular automaton is a modification of a 1D three neighbors elementary CA. Such modification is in the mapping function of the CA and the next state in time $t+1$ of the central cell c_i is defined by a probability p to be 1 and a probability $1-p$ to be 0 for each of the eight different neighborhood patterns this CA has. Formally, probability p is represented by



0	0	0	0	1	0	1	1	0	1	1	1	0	1	1	1

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	2	1	0	0	0	0	0	0	0	0	0	0	0	0	0	4
1	4	2	1	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	4	2	1	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	4	2	1	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	4	2	1	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	4	2	1	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	4	2	1	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	4	2	1	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	4	2	1	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	4	2	1	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	4	2	1	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	4	2	1	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0	4	2	1	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	4	2	1	0
14	0	0	0	0	0	0	0	0	0	0	0	0	0	4	2	1
15	1	0	0	0	0	0	0	0	0	0	0	0	0	0	4	2

	0
0	0
1	0
2	0
3	0
4	1
5	0
6	1
7	1
8	0
9	1
10	1
11	1
12	0
13	1
14	1
15	1

	0
0	4
1	0
2	0
3	1
4	2
5	5
6	3
7	6
8	5
9	3
10	7
11	6
12	5
13	3
14	7
15	6

Fig. 2 Example of using matrix multiplication for computing a 1D elementary cellular automaton with 16 cells and wrapped grid. **a** Example of the grid of cells with states. State 0 means dead or non-occupied cell and state 1 stands for alive or occupied cell. **b** Indices of the cells and 3-neighbors pattern neighborhood of 1D CA where thick border means the current cell and thin border means the neighbors. **c** Illustration of matrix multiplication between adjacency matrix of the 1D CA and the state vector of the 1D CA, resulting in a vector that contains the pattern code of the neighborhood for each cell. Important to consider that an alive cell counts itself as an alive neighbor and that is why the diagonal of the adjacency matrix is fulfilled with weight 2

$$p = P(c_{i,t+1} = 1 | N(c_{i,t})), \tag{3}$$

where the neighborhood pattern $N(c_{i,t})$ is denoted as

$$N(c_{i,t}) = (c_{i-1,t}, c_{i,t}, c_{i+1,t}). \tag{4}$$

The second stochastic dynamical system that we evolve is based on random Boolean networks (RBNs). Basically, this is a modification of our stochastic cellular automata, but with the connectivity between the cells being random.

Our third and last stochastic dynamical system is based on echo state networks (ESNs). As its activation function, we use the sigmoid function denoted as

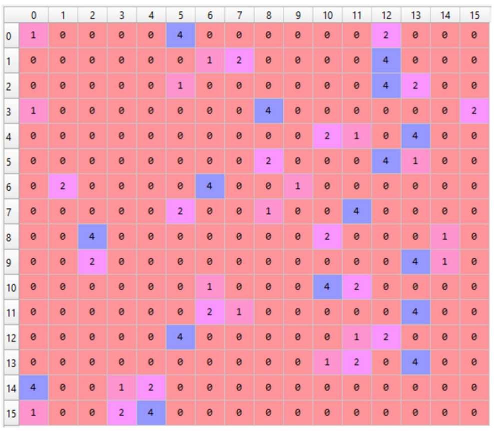
$$\mathrm{sigmoid}(x) = \frac{1}{1 + e^{-x}}. \tag{5}$$

Since our echo state network is stochastic, the probability p_{ESN} of next state being 1 is calculated by the sigmoid function in (5). This is given formally by

$$p_{ESN} = P(c_{i,t+1} = 1) = \mathrm{sigmoid}(\mathbf{A} \cdot \mathbf{c}_t). \tag{6}$$

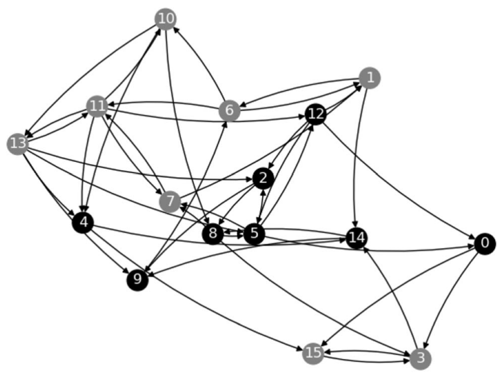
Evolution through genetic algorithm

The evolution towards criticality is performed by a genetic algorithm. As described in the previous section, three different stochastic dynamical systems are evolved: CA, RBN and ESN. The genotype (or genetic code) for CA and RBN is the same. It contains one probability (value between 0.0 and 1.0) for each of the eight possible neighborhood configurations (three binary neighbors). The genome of the ESN consists of six values denoting mean and standard deviation of the weights of the positive connections ($mean_+$ and std_+), mean and standard deviation of the negative connections ($mean_-$ and std_-), probability of positive connections ($prob+$), and *sparsity*. The range of $mean_+$ and $mean_-$ is between 0.2 and 4.0, the values of std_+ and std_- are determined by $mean_+$ and $mean_-$, and their genes $geneStd_+$ and $geneStd_-$ (values between 0.0 and 1.0). The equations for std_+ and std_- are



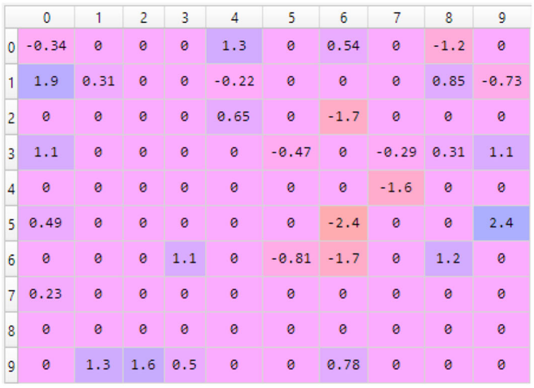
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	1	0	0	0	0	4	0	0	0	0	0	0	2	0	0	0
1	0	0	0	0	0	0	1	2	0	0	0	0	4	0	0	0
2	0	0	0	0	0	1	0	0	0	0	0	0	4	2	0	0
3	1	0	0	0	0	0	0	0	4	0	0	0	0	0	0	2
4	0	0	0	0	0	0	0	0	0	0	2	1	0	4	0	0
5	0	0	0	0	0	0	0	0	2	0	0	0	4	1	0	0
6	0	2	0	0	0	0	4	0	0	1	0	0	0	0	0	0
7	0	0	0	0	0	2	0	0	1	0	0	4	0	0	0	0
8	0	0	4	0	0	0	0	0	0	0	2	0	0	0	1	0
9	0	0	2	0	0	0	0	0	0	0	0	0	0	4	1	0
10	0	0	0	0	0	0	1	0	0	0	4	2	0	0	0	0
11	0	0	0	0	0	0	2	1	0	0	0	0	0	4	0	0
12	0	0	0	0	0	4	0	0	0	0	0	1	2	0	0	0
13	0	0	0	0	0	0	0	0	0	0	1	2	0	4	0	0
14	4	0	0	1	2	0	0	0	0	0	0	0	0	0	0	0
15	1	0	0	2	4	0	0	0	0	0	0	0	0	0	0	0

(a)



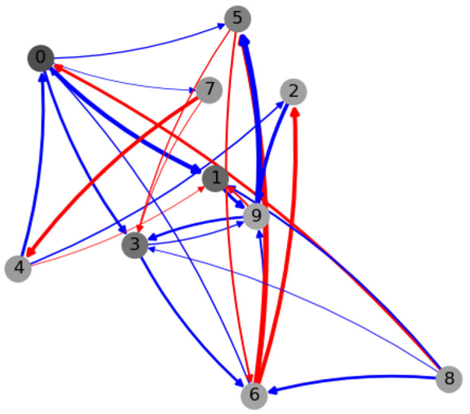
(b)

Fig. 3 Example of a weighted adjacency matrix and graph for a random Boolean network with 16 cells and neighborhood of 3 cells. Self-connections are not shown in the graph



	0	1	2	3	4	5	6	7	8	9
0	-0.34	0	0	0	1.3	0	0.54	0	-1.2	0
1	1.9	0.31	0	0	-0.22	0	0	0	0.85	-0.73
2	0	0	0	0	0.65	0	-1.7	0	0	0
3	1.1	0	0	0	0	-0.47	0	-0.29	0.31	1.1
4	0	0	0	0	0	0	0	-1.6	0	0
5	0.49	0	0	0	0	0	-2.4	0	0	2.4
6	0	0	0	1.1	0	-0.81	-1.7	0	1.2	0
7	0.23	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0	0
9	0	1.3	1.6	0.5	0	0	0.78	0	0	0

(a)



(b)

Fig. 4 Example of a weighted adjacency matrix and graph for an echo state network with 10 cells or neurons. Red edges mean negative connections and blue edges mean positive connections. The thickness of the edges indicates the weight values. Self-connections are not shown in the graph

$$\text{std}_+ = 0.2 \times \text{mean}_+ * \text{geneStd}_+, \tag{7}$$

$$\text{std}_- = 0.2 \times \text{mean}_- * \text{geneStd}_-. \tag{8}$$

The standard deviation values have a minimum of 0.0 and a maximum of 20% of their corresponding mean. Such a maximum value for the standard deviation reduces the chances of sampling negative weights from the positive weight normal distribution, and vice-versa. However, in case this occurs, the absolute function is applied.

The fitness function which guides the stochastic dynamical systems towards criticality mainly verifies whether the probability distributions of avalanche size and duration follow a power-law distribution. The avalanche size and duration are acquired by the cluster size of identical states, which means the number of repetitions of a state that happened consecutively without the interruption of another state. The avalanche size stands for the clusters in the states in the same time-step and the avalanche duration consists of the clusters in the same cell through the time-steps of the simulation. The power-law distribution verification of the probability distributions of avalanche size and duration can be done in several ways. In our task, evolution is based on the verification of linearity in a log-log plot and the model comparison between power-law and exponential by the log-likelihood ratio (Clauset et al. 2009). The model comparison is an addition to the previous version of the fitness function for criticality in (Pontes-Filho et al. 2019a), which facilitates the convergence towards such a goal. After the evolution is completed, we test the best genome or individual with goodness-of-fit tests based on the Kolmogorov-Smirnov (KS) statistic (Clauset et al. 2009). To do that, the *p*-value of goodness-of-fit test is calculated using 1000 randomly generated data with 10,000 samples applying the power-law model estimated by maximum likelihood estimation method with minimum *x* of the distribution fixed to 1. The *p*-value measures the percentage of the KS statistic of the generated data when it is greater (worse) than the KS statistic of the empirical distribution. Therefore, a *p*-value of 1.0 or 100% is the best possible value and, to be accepted as power-law, the *p*-value must be greater than 0.1 (Clauset et al. 2009). The fitness function does not have goodness-of-fit test because it is computationally intensive. In our code, the log-likelihood ratio, generation of data from power-law model, and maximum likelihood estimation method are imported from the powerlaw Python library (Alstott et al. 2014).

The fitness function, used during evolution to calculate the genome's fitness score, estimates the power-law model of the four distributions (avalanche size and duration for the state 0 and 1) acquired from the simulation of the stochastic binary dynamical system produced by the genome. The simulation runs 1,000 time-steps of a system with 1000 cells. The power-law model estimation is performed by linear fitting of the first 10 points of the log-log plot using least squares regression, which was verified to be unbiased and gives a fast and acceptable estimation of the slope of the power-law distribution (Goldstein et al. 2004). Their power-law models and empirical probability distributions are subsequently compared with the KS statistic and coefficient of determination (Wright 1921). The advantage of using the KS statistic with a model estimated by a linear 10-points fitting is that it reports a large error when the empirical distribution does not follow a power-law distribution. Another objective in the fitness function is the number of non-zero bins of size one in the raw histogram (empirical probability distribution). The number of non-zero bins is then normalized by dividing it with the maximum number of bins, which is 1000 for our case because 1000 cells are simulated through 1000 time-steps. Another objective is the percentage of unique states during the simulation (value between 0.0 and 1.0). In summary, the fitness function has scores calculated from the four probability distributions, which are the normalized number of non-zero bins *bin*; coefficient of determination R^2 of complete linear fitting; and KS statistic *D*. All these values are vectors of four elements. The fitness score *s* for those objectives is then calculated by the following equations:

$$bin_s = \tanh(5 * (0.9 * \max(bin) + 0.1 * \mathrm{mean}(bin))), \tag{9}$$

$$R_s^2 = \mathrm{mean}(R^2), \tag{10}$$

$$D_s = \exp(-(0.9 * \min(D) + 0.1 * \mathrm{mean}(D))). \tag{11}$$

The fitness score which is based on the simulation result is the percentage of unique states, which is denoted by

$$unique_s = \frac{\#uniqueStates}{\#timesteps}. \tag{12}$$

The Eqs. (9)–(12) are all objective values for calculating the temporary fitness score s_{temp}. Those values are real numbers between zero and one. Some important scores are squared, such as R_s^2 and D_s. The following equation denotes how the temporary fitness score s_{temp} is calculated:

$$s_{temp} = bin_s + (R_s^2)^2 + (D_s)^2 + unique_s. \tag{13}$$

The final fitness score includes the log-likelihood ratio which compares the power-law model with the exponential model for estimating the probability distribution. This process is computationally intensive. Therefore, such a score is only computed when the temporary fitness score s_{temp} reaches a certain value. If the s_{temp} is greater than this threshold value of 3.5, then the log-likelihood ratio is calculated for the four distributions and stored in the vector *l*. The log-likelihood ratio which is not trustworthy (*p*-value of ratio greater or equal to 0.1) are ignored (set as zero). The score for the log-likelihood ratio l_s is then calculated by

$$l_s = \mathrm{sigmoid}(10^{-2} * (0.9 * \max(l) + 0.1 * \mathrm{mean}(l))). \tag{14}$$

After describing all the objectives and their scores of our fitness function, the final equation is

$$s = \begin{cases} s_{temp} + l_s, & s_{temp} > 3.5 \\ s_{temp}, & \text{otherwise.} \end{cases} \tag{15}$$

The configuration of the genetic algorithm consists of 40 individuals evolving through 100 generations. We run the genetic algorithm five times for each of the three dynamical systems. The goal of the genetic algorithm is to maximize the fitness score. The selection of two parents is done by deterministic tournament selection of two individuals (Goldberg and Deb 1991), which means that all individuals are assigned for the tournaments. Afterwards, the crossover between the genomes of the selected parents may occur with probability 0.8, and then each gene can be exchanged with probability 0.5. After that, a mutation can modify a gene with probability 0.1. This mutation adds a random value from a normal distribution with mean and standard deviation equal to 0 and 0.2, respectively. The mating process of the two parents produces an offspring of two new individuals who replace the parents in the next generation.

Experimental results

The results of the methods described for a general framework for dynamical systems are described and explained in this section. The results of the genetic algorithm for criticality in three stochastic dynamical systems are also described and explained.

Results of general framework

Figure 1 shows the result of Algorithm 2. It describes a wrapped 2D CA (similar to Game of Life but with a lower number of neighbors) and shows the resulting adjacency matrix. Figure 1a illustrates the desired two-dimensional CA with 16 cells (i.e., $widthCA = 4$ and $heightCA = 4$). Figure 1b presents the von Neumann neighborhood without considering the center cell (Toffoli and Margolus 1987) which is used for counting the number of "alive" neighbors (the connection weights are only zero and one, and defined by **Neighborhood** argument of Algorithm 2). It also shows the index distribution of the CA whose order is preserved after flattening it to a column vector. Figure 1c contains the generated adjacency matrix of Algorithm 2 for the described 2D CA. Figure 1b shows an example of a central cell with its neighbors, the index of this central cell is 5 and the row index 5 in the adjacency matrix of Fig. 1c presents the same neighbor indices, i.e., 1, 4, 6 and 9. Since this is a symmetric matrix, the columns have the same connectivity of the rows. This implies that the neighborhood of a cell considers the cell itself as a neighbor. Therefore, the connections are bidirectional and the adjacency matrix represents an undirected graph. The wrapping effect is also observable. For example, the neighbors of the cell index 0 are 1, 3, 4 and 12. So the neighbors 3 and 12 are the ones that the wrapped grid allowed to exist for cell index 0.

Figure 2 contains the result of Algorithm 1 together with (2). It illustrates a wrapped elementary CA and its generated weighted adjacency matrix. Figure 2a shows the appearance of the desired elementary CA with 16 cells ($widthCA = 16$). Figure 2b describes its 3-neighborhood pattern and the indices of the cells. Figure 2c shows the result of Algorithm 1 with the neighborhood calculated by (2) for pattern matching in the activation function. In Fig. 2c, we can verify that the left neighbor has weight equal to 4 (or 2^2 for the most significant bit), central cell weight is 2 (or 2^1) and the right neighbor weight is 1 (or 2^0 for the least significant bit) as defined by (2). Since the CA is wrapped, we can notice in row index 0 of the adjacency matrix in Fig. 2c that the left neighbor of cell 0 is cell 15, and in row index 15 that the right neighbor of cell 15 is cell 0.

Figure 3 sets out the result of (2). The neighborhood is defined as n-ary string for the purpose of identifying the states of each neighbor. The neighbors of a cell are selected randomly and are represented in the matrix row of the cell's index. Therefore, the neighbor identifiers, which are in this case 1, 2 and 4, are assigned to their corresponding neighbor.

Results of evolving dynamical systems towards criticality

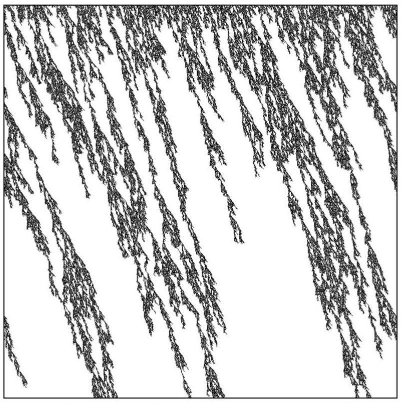
After five independent runs of the CA evolution, the best genome solutions turn out to be unstable, i.e., the test score of the best genome differs significantly when compared to the score obtained during evolution. For this reason, the 2nd best solution is selected, as its test score shows

Table 2 Selected 2nd best CA in fitness score

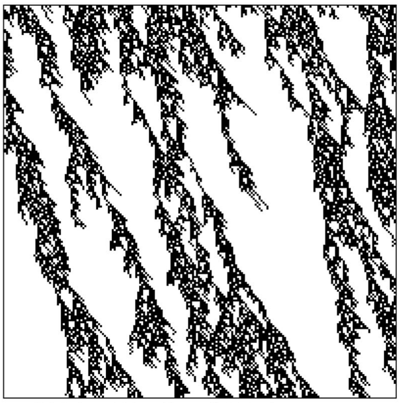
Neighborhood $N(c_{i,t})$	Probability p
(0,0,0)	0.394221
(0,0,1)	0.094721
(0,1,0)	0.239492
(0,1,1)	0.408455
(1,0,0)	0.000000
(1,0,1)	0.730203
(1,1,0)	0.915034
(1,1,1)	1.000000

Table 3 Fitness score of the selected 2nd best CA. Testing simulations were performed 5 times and "std." stands for standard deviation. Numbers are rounded to three decimal places

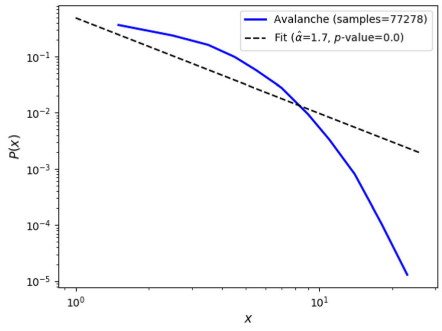
Objective	Evolution score	Test score mean	Test score std.
R_s^2	0.870	0.866	0.006
D_s	0.961	0.961	0.003
bin_s	0.966	0.980	0.007
$unique_s$	1.000	1.000	0.000
l_s	0.728	0.733	0.016
s	**4.376**	**4.387**	**0.015**



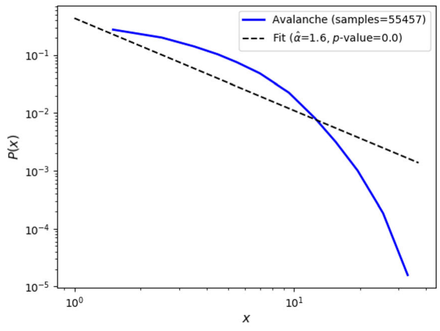
(a) Entire simulation of 1,000 cells and 1,000 time-steps



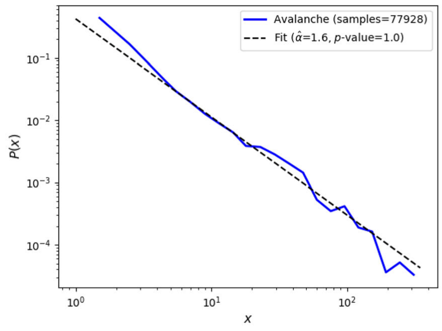
(b) First 200 cells and 200 time-steps



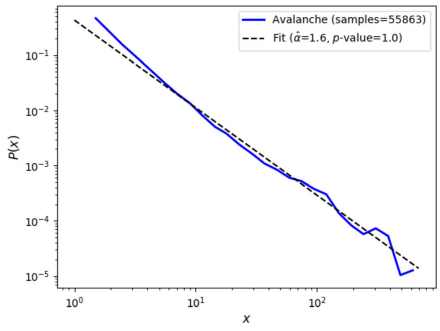
(c) Avalanche size of state 0



(d) Avalanche duration of state 0



(e) Avalanche size of state 1



(f) Avalanche duration of state 1

Fig. 5 Test sample of the 2nd best evolved stochastic elementary CA of 1,000 cells (horizontal axis) randomly initialized with wrapped boundaries and run through 1,000 time-steps (vertical axis), and its avalanche size and duration of the two states 0 (black) and 1 (white). Fitness score of this simulation is 4.383

stable results. The genome of the stable solution is presented in Table 2. Its fitness score and all objective scores during evolution and testing are in Table 3. It can be observed that the CA results are stable because of the low standard deviation of the scores in the five testing executions. This is further supported by the mean test score being larger than the score during evolution. Fig. 5 contains the image produced by the entire simulation, by the first 200 cells and 200 time-steps, and by the four probability distributions with their corresponding power-law model estimated by maximum log-likelihood and p-value of the goodness-of-fit test. The empirical probability distributions (depicted in Figs. 5c–f) which fit to a power-law model are the probability distributions of avalanche size and duration of state 1 (Figs. 5e and 5f). This can be concluded quantitatively by the p-values of their goodness-of-fit test being equal to 1.0, which to be considered a power-law distribution p-value must be greater than 0.1 (Clauset et al. 2009). Moreover, the large number of samples confirms that these p-values are reliable; and qualitatively by the similarity of their power-law estimated models (black dashed line) and the empirical distributions (blue solid line). Therefore, we can conclude that the presented CA shows criticality for state 1.

Repeating the same procedure used for CA, the RBN's 1st best individual presented a high score as the 2nd best CA score, but the 1st best RBN is unstable. The following best individuals are also showing instability. Hence, we keep the selection of the 1st best individual. Table 4 contains the genome of the selected RBN. Table 5 has the scores acquired during evolution and the mean and standard deviation of the five test runs. Figure 6 illustrates the simulation of the RBN and their avalanche distributions. It can be noted that none of the distributions qualitatively resembles a power-law, but Fig. 6c shows the distribution of avalanche size of state 0 which has a p-value of goodness-of-fit test equal to 1.0 which means that it is classified as power-law according to this evaluation method. Nevertheless, if we consider that such RBN does not achieve criticality, we can hypothesize that the random connections may be a bottleneck to achieving this behavioral regime while, with a regular grid, CA more easily achieved a critical behavior through its evolution.

The ESN results are presented in Table 6, Table 7, and Fig. 7. The 1st best ESN was found to be unstable as the 1st best CA. Therefore, the selected genome is the 2nd best which presents stable results. The CA and ESN's selected best individuals possess two distributions which are considered power-laws by the p-value of goodness-of-fit test. However, the ESN's avalanche distributions with p-value equal to 1.0 are the avalanche duration of state 0 and 1. This means that avalanches that present criticality do not occur within the states through the simulation. The criticality occurs only by combining the cluster sizes of each of the cells in the system during the simulation.

We consider that the evolved stochastic dynamical system achieved criticality when at least one of the probability distributions of the avalanche size and duration is a power-law distribution. That is, quantitatively evaluated by the p-value of the goodness-of-fit test. Table 8 contains the mean and standard deviation of the p-value of the four avalanche distributions. Through this result, we can affirm that two out of the four presented distributions for the CA and ESN show a power-law distribution, i.e., at criticality. The presented results also show that the tested RBN possesses only one avalanche distribution which can be considered as a power-law; the avalanche size distribution of state 0. Moreover, the p-value of this distribution of RBN is not as stable as the two critical avalanche distributions of CA and ESN with mean equaling 1.0 and standard deviation equaling 0.0.

Table 4 Selected 1st best RBN in fitness score

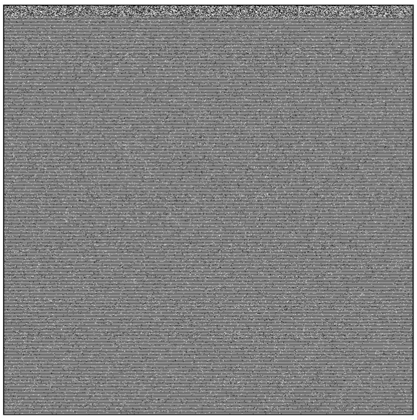
Neighborhood $N(c_{i,t})$	Probability p
(0,0,0)	1.000000
(0,0,1)	0.844143
(0,1,0)	0.950141
(0,1,1)	0.314001
(1,0,0)	0.527704
(1,0,1)	0.314433
(1,1,0)	0.109056
(1,1,1)	0.015699

Table 5 Fitness score of the selected 1st best RBN. Testing simulations were performed 5 times and "std." stands for standard deviation. Numbers are rounded to three decimal places

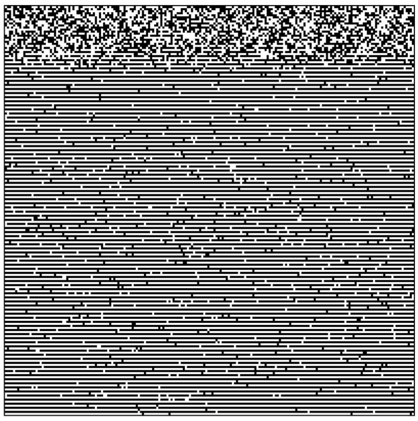
Objective	Evolution score	Test score mean	Test score std.
R_s^2	0.886	0.905	0.002
D_s	0.953	0.867	0.050
$bins_s$	0.867	0.864	0.007
$unique_s$	1.000	1.000	0.000
l_s	0.706	0.145	0.291
s	**4.266**	**3.583**	**0.353**

Discussion

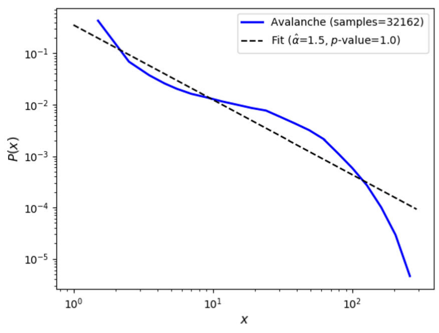
The results of the evolution of the three stochastic dynamical systems show the potential of such systems to produce criticality. Evaluating these systems, we can



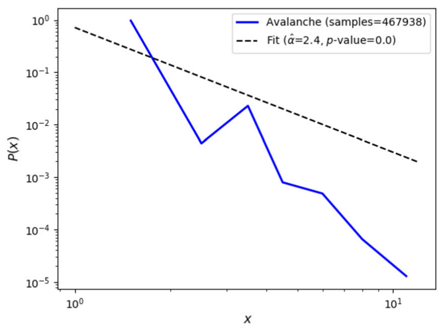
(a) Entire simulation of 1,000 cells and 1,000 time-steps



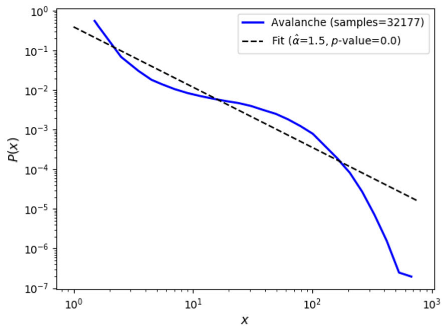
(b) First 200 cells and 200 time-steps



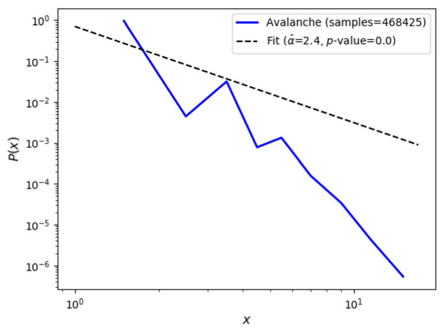
(c) Avalanche size of state 0



(d) Avalanche duration of state 0



(e) Avalanche size of state 1



(f) Avalanche duration of state 1

Fig. 6 Test sample of the 1st best evolved stochastic RBN of 1000 cells (horizontal axis) randomly initialized and run through 1000 time-steps (vertical axis), and its avalanche size and duration of the two states 0 (black) and 1 (white). Fitness score of this simulation is 3.315

Table 6 Selected 2nd best ESN in fitness score

Genome	Value
mean$_+$	4.000000
std$_+$	0.800000
mean$_-$	0.100000
std$_-$	0.007792
prob+	0.064934
sparsity	0.963955

Table 7 Fitness score of the selected 2nd best ESN. Testing simulations were performed 5 times and “std.” stands for standard deviation. Numbers are rounded to three decimal places

Objective	Evolution score	Test score mean	Test score std.
R_s^2	0.891	0.891	0.006
D_s	0.903	0.885	0.038
$bins_s$	0.968	0.965	0.004
$unique_s$	1.000	1.000	0.000
l_s	0.613	0.479	0.239
s	**4.190**	**4.024**	**0.282**

deduce that the stochastic cellular automaton is the system that can become critical most easily. This is followed by the stochastic echo state network, which in our results presented an unexpected behavior where the only avalanche distributions that can be considered critical are the two avalanche duration distributions. This result is unexpected if compared to the presented CA, which presents only one state (state 1) as critical in both avalanche size and duration. The stochastic random Boolean network is very similar to the stochastic CA, with the difference that the connectivity is randomized instead of regular. Such modification may make it more difficult to evolve the RBN into a critical system behavior. The RBN only shows a single critical avalanche distribution and is not stable like the two critical avalanche distributions of CA and ESN.

Ongoing and future applications with EvoDynamic

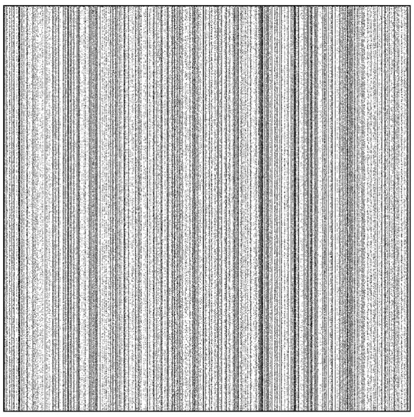
The generalization of representations for different dynamical systems presented in this work is beneficial for the further development of the EvoDynamic framework. Cellular automata, random Boolean networks, and echo state networks are already implemented in our Python library. The implementation of the other described dynamical systems in the EvoDynamic framework is ongoing. In addition, the EvoDynamic framework will incorporate the possibility to evolve the connectivity, the update rules and the learning rules of the dynamical systems, in order to allow the dynamical systems to be used efficiently for reservoir computing, as well as for physical substrate modeling. The introduced general representation facilitates the evolution of such systems and models through methods that measure the quality of a reservoir system or the similarity to a given input dataset. The following subsection will further document an additional method under development, which can be used to assess the quality of a dynamical system model or substrate for reservoir computing.

State trajectory

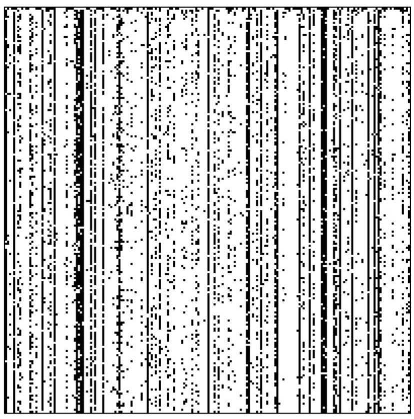
A method that can guide dynamical systems’ evolutionary search is the state trajectory. This method can be used to cluster similar states for model abstraction and to measure the quality of the reservoir. For this purpose, a graph can be generated and analyzed by searching for attractors and cycles in the obtained state space. For visualization of the state trajectory, we apply principal component analysis (PCA) to reduce the dimensionality of the states considering the entire dynamical system simulation (each timestep produces a sample for PCA). An example of the produced visualization is depicted in Fig. 8, where every produced state is shown as a state transition diagram. The chosen dynamical system shown in the Figure is a CA using Conway’s Game of Life’s rules with 5 x 5 cells and wrapped boundaries. The CA is initialized with a glider configuration as the initial state (Fig. 8a) and, subsequently, the CA cycles over 20 unique states, as illustrated in the state transition diagram in Fig. 8l.

Conclusion

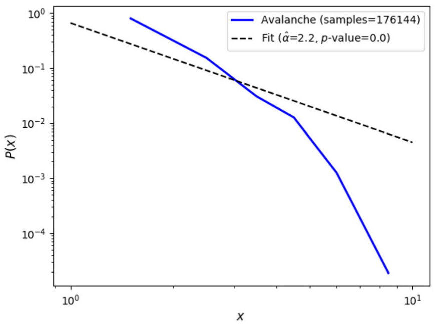
In this work, a general framework for simulating dynamical systems is described, which utilizes the computation of artificial neural networks as a general method for executing different dynamical systems. The presented framework, called EvoDynamic, is built on the Tensorflow deep learning library, which allows better performance and parallelization while keeping a common general representation based on operations on sparse tensors. The application of this framework is used in the work herein to evolve three different dynamical systems, i.e., cellular automata, random Boolean networks, and echo state networks, towards criticality. The presented results are promising for CA and ESN evolution, while further analysis and experiments are required to confirm critical behavior in the evolved RBNs. As future work, our goal is to evolve dynamical systems towards self-organized criticality, i.e., a dynamical system that self-organizes into a critical state



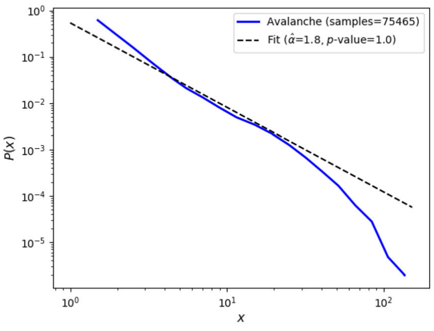
(a) Entire simulation of 1,000 cells and 1,000 time-steps



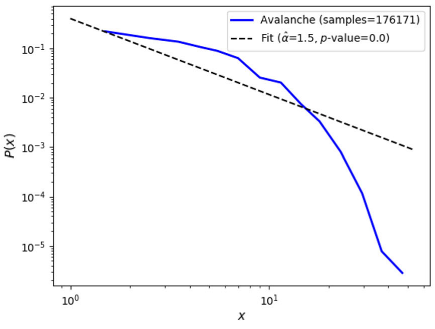
(b) First 200 cells and 200 time-steps



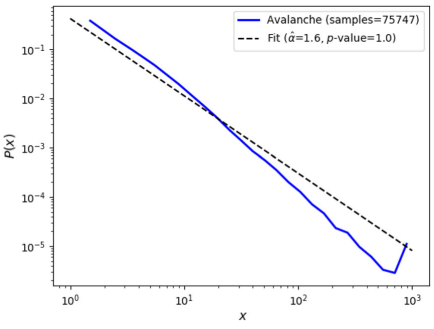
(c) Avalanche size of state 0



(d) Avalanche duration of state 0



(e) Avalanche size of state 1

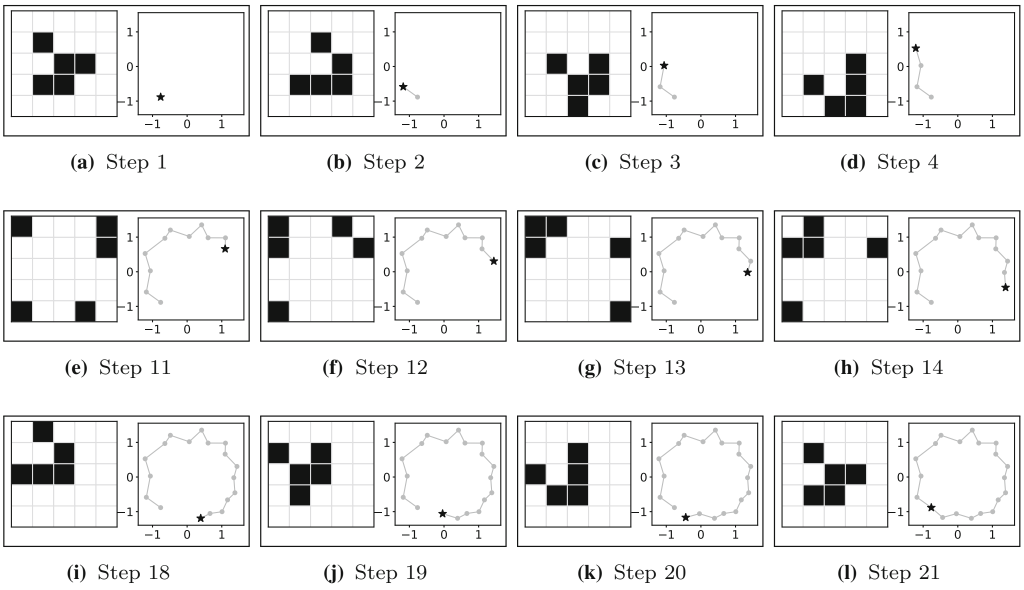


(f) Avalanche duration of state 1

Fig. 7 Test sample of the 2nd best evolved stochastic ESN of 1000 cells (horizontal axis) randomly initialized and run through 1,000 time-steps (vertical axis), and its avalanche size and duration of the two states 0 (black) and 1 (white). Fitness score of this simulation is 4.158

Table 8 Goodness-of-fit test of the three evolved stochastic dynamical systems. Avalanche size (AS) and avalanche duration (AD) are followed by the state from which they were calculated. Testing simulations were performed 5 times and p-values are denoted as "mean ± standard deviation". The p-values in bold are the ones that are considered a power-law distribution. So, p-value > 0.1

System	p-value of AS of state 0	p-value of AD of state 0	p-value of AS of state 1	p-value of AD of state 1
CA	0.0 ± 0.0	0.0 ± 0.0	**1.0 ± 0.0**	**1.0 ± 0.0**
RBN	**0.969 ± 0.021**	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
ESN	0.0 ± 0.0	**1.0 ± 0.0**	0.0 ± 0.0	**1.0 ± 0.0**


(a) Step 1 **(b)** Step 2 **(c)** Step 3 **(d)** Step 4

(e) Step 11 **(f)** Step 12 **(g)** Step 13 **(h)** Step 14

(i) Step 18 **(j)** Step 19 **(k)** Step 20 **(l)** Step 21

Fig. 8 States of Conway's Game of Life in a 5 x 5 wrapped lattice alongside their PCA-transformed state transition diagrams of the two first principal components. **a** Initial state is a glider. **a–d** Four first steps in this CA. **e–h** Four intermediate steps in this CA while reaching the wrapped border. **i–l** Four last steps in this CA before repeating the initial state and closing a cycle

without the need to tune control parameters. Ongoing and future implementations of EvoDynamic are presented and discussed, such as the visualization and usage of state trajectories, as well as the possibility of physical substrate modeling. EvoDynamic is an open-source framework currently under development that primarily targets applications in reservoir computing and artificial intelligence. We envision that the generalization and parallelization of the described dynamical systems will enable our Python library to be widely used by the research community.

Acknowledgements Open Access funding provided by OsloMet - Oslo Metropolitan University. We thank Kristine Heiney for thoughtful discussions about self-organized criticality. This work was supported by Norwegian Research Council SOCRATES project (Grant Number 270961).

Compliance with ethical standards

Conflict of interest The authors declare that they have no conflict of interest.

References

Aaser P, Knudsen M, Ramstad O.H, van de Wijdeven R, Nichele S, Sandvig I, Tufte G, Stefan Bauer U, Halaas Ø, Hendseth S, Sandvig A, Valderhaug V (2017) Towards making a cyborg: A closed-loop reservoir-neuro system. The 2018 Conference on Artificial Life: A Hybrid of the European Conference on Artificial Life (ECAL) and the International Conference on the Synthesis and Simulation of Living Systems (ALIFE) (29), 430–437. https://doi.org/10.1162/isal_a_072.https://www.mitpressjournals.org/doi/abs/10.1162/isal_a_072

Abadi M, Barham P, Chen J, Chen Z, Davis A, Dean J, Devin M, Ghemawat S, Irving G, Isard M, Kudlur M, Levenberg J, Monga R, Moore S, Murray D.G, Steiner B, Tucker P, Vasudevan V, Warden P, Wicke M, Yu Y, Zheng X (2016) Tensorflow: A system for large-scale machine learning. In: 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16), pp. 265–283. USENIX Association, Savannah, GA. https://www.usenix.org/conference/osdi16/technical-sessions/presentation/abadi

Alstott J, Bullmore E, Plenz D (2014) Powerlaw: a python package for analysis of heavy-tailed distributions. PLoS ONE 9(1):1–11. https://doi.org/10.1371/journal.pone.0085777

Baetens JM, Van der Meeren W, De Baets B (2016) On the dynamics of stochastic elementary cellular automata. J Cellular Automata 12:63–80

Bailey J.A (2010) Towards the neurocomputer: an investigation of vhdl neuron models. Ph.D. thesis, University of Southampton

Bak P, Tang C, Wiesenfeld K (1987) Self-organized criticality: an explanation of the 1/f noise. Phys. Rev. Lett. 59:381–384. https://doi.org/10.1103/PhysRevLett.59.381

Broersma H, Miller JF, Nichele S (2017) Computational matter: evolving computational functions in nanoscale materials. Springer, Cham, pp 397–428

Chow SN, Mallet-Paret J, Van Vleck ES (1996) Dynamics of lattice differential equations. Int J Bifurcation Chaos 6(09):1605–1621

Clauset A, Shalizi CR, Newman ME (2009) Power-law distributions in empirical data. SIAM Rev 51(4):661–703

Gershenson C (2004) Introduction to random boolean networks. arXiv preprint nlin/0408006

Goldberg D.E, Deb K (1991) A comparative analysis of selection schemes used in genetic algorithms. In: Foundations of genetic algorithms, Elsevier, vol 1, pp 69–93

Goldstein ML, Morris SA, Yen GG (2004) Problems with fitting to the power-law distribution. European Phys J B-Condens Matter Complex Syst 41(2):255–258

Heiney K, Ramstad O.H, Sandvig I, Sandvig A, Nichele S (2019) Assessment and manipulation of the computational capacity of in vitro neuronal networks through criticality in neuronal avalanches. arXiv preprint arXiv:1907.13118

Jaeger H, Haas H (2004) Harnessing nonlinearity: predicting chaotic systems and saving energy in wireless communication. Science 304(5667):78–80.https://doi.org/10.1126/science.1091277

Jensen J.H, Folven E, Tufte G (2018) Computation in artificial spin ice. The 2018 Conference on Artificial Life: A Hybrid of the European Conference on Artificial Life (ECAL) and the International Conference on the Synthesis and Simulation of Living Systems (ALIFE) (30), 15–22. https://doi.org/10.1162/isal_a_00011.https://www.mitpressjournals.org/doi/abs/10.1162/isal_a_00011

Kaneko K (1992) Overview of coupled map lattices. Chaos An Interdisciplinary J Nonlinear Sci 2(3):279–282

Konkoli Z, Nichele S, Dale M, Stepney S (2018) Reservoir computing with computational matter. Springer, Cham, pp 263–293

Langton CG (1990) Computation at the edge of chaos: phase transitions and emergent computation. Phys D Nonlinear Phenomena 42(1):12–37https://doi.org/10.1016/0167-2789(90)90064-V

Larter R, Speelman B, Worth RM (1999) A coupled ordinary differential equation lattice model for the simulation of epileptic seizures. Chaos An Interdisciplinary J Nonlinear Sci 9(3):795–804. https://doi.org/10.1063/1.166453

LeCun Y, Bengio Y, Hinton G (2015) Deep learning. Nature 521(7553):436–444

Maass W, Markram H (2004) On the computational power of circuits of spiking neurons. J Comput Syst Sci 69(4):593–616. https://doi.org/10.1016/j.jcss.2004.04.001

Markram H, Meier K, Lippert T, Grillner S, Frackowiak R, Dehaene S, Knoll A, Sompolinsky H, Verstreken K, DeFelipe J, Grant S, Changeux JP, Saria A (2011) Introducing the human brain project. Procedia Comput Sci 7:39–42

McAfee A, Brynjolfsson E, Davenport TH, Patil D, Barton D (2012) Big data: the management revolution. Harvard Bus Rev 90(10):60–68

Nichele S, Gundersen MS (2017) Reservoir computing using nonuniform binary cellular automata. Complex Syst 26(3):225–245https://doi.org/10.25088/complexsystems.26.3.225

Nichele S, Molund A (2017) Deep learning with cellular automaton-based reservoir computing. Complex Syst 26(4):319–339https://doi.org/10.25088/complexsystems.26.4.319

Nichele S, Tufte G (2010) Trajectories and attractors as specification for the evolution of behaviour in cellular automata. In: IEEE Congress on evolutionary computation, pp. 1–8. https://doi.org/10.1109/CEC.2010.5586115

Nichele S, Tufte G (2012) Genome parameters as information to forecast emergent developmental behaviors. In: Durand-Lose J, Jonoska N (eds) Unconventional computation and natural computation. Springer, Berlin, pp 186–197

Nichele S, Farstad SS, Tufte G (2017) Universality of evolved cellular automata in-materio. Int J Unconvent Comput 13(1):1-34

Oussous A, Benjelloun FZ, Lahcen AA, Belfkih S (2018) Big data technologies: a survey. J King Saud Univ Comput Information Sci 30(4):431–448. https://doi.org/10.1016/j.jksuci.2017.06.001

Pontes-Filho S, Lind P, Yazidi A, Zhang J, Hammer H, Mello GB, Sandvig I, Tufte G, Nichele S (2019a) Evodynamic: a framework for the evolution of generally represented dynamical systems and its application to self-organized criticality. Tech. rep, EasyChair

Pontes-Filho S, Yazidi A, Zhang J, Hammer H, Mello G.B, Sandvig I, Tufte G, Nichele S (2019b) A general representation of dynamical systems for reservoir computing. In: Workshop on Novel Substrates and Models for the Emergence of Developmental, Learning and Cognitive Capabilities

Rendell P (2002) Turing universality of the game of life. Springer, London, pp 513–539https://doi.org/10.1007/978-1-4471-0129-1_18

Schrauwen B, Verstraeten D, Van Campenhout J (2007) An overview of reservoir computing: theory, applications and implementations. In: Proceedings of the 15th European Symposium on Artificial Neural Networks. pp. 471-482

SOCRATES Self-Organizing Computational substRATES. https://www.ntnu.edu/socrates

Subramoney A, Scherr F, Maass W (2019) Reservoirs learn to learn. arXiv preprint arXiv:1909.07486

Tanaka G, Yamane T, Hroux JB, Nakane R, Kanazawa N, Takeda S, Numata H, Nakano D, Hirose A (2019) Recent advances in physical reservoir computing: a review. Neural Netw 115:100–123https://doi.org/10.1016/j.neunet.2019.03.005

TensorFlow: tf.sparse.sparse_dense_matmul | tensorflow core r1.14 | tensorflow. https://www.tensorflow.org/api_docs/python/tf/sparse/sparse_dense_matmul

Tetzlaff C, Okujeni S, Egert U, Wörgötter F, Butz M (2010) Self-organized criticality in developing neuronal networks. PLoS Comput Biol 6(12):e1001013

Toffoli T, Margolus N (1987) Cellular automata machines: a new environment for modeling. MIT press, Cambridge

Wolfram S (2002) A new kind of science, vol 5. Wolfram media, Champaign

Wright S (1921) Correlation and causation. J Agric Res 20:557–580

Yada Y, Mita T, Sanada A, Yano R, Kanzaki R, Bakkum DJ, Hierlemann A, Takahashi H (2017) Development of neural population activity toward self-organized criticality. Neuroscience 343:55–65

Paper A.3

Assessing the robustness of critical behavior in stochastic cellular automata
(Pontes-Filho et al., 2022)

Author(s):
Sidney Pontes-Filho, Pedro Lind and Stefano Nichele

Published in journal:
Physica D: Nonlinear Phenomena, vol. 441, p. 133507, 2022

Paper A.3
Pontes-Filho et al. (2022)

Contents lists available at ScienceDirect

Physica D

journal homepage: www.elsevier.com/locate/physd

Assessing the robustness of critical behavior in stochastic cellular automata

Sidney Pontes-Filho [a,b,*], Pedro G. Lind [a,c,d], Stefano Nichele [a,c,d,e,f]

[a] *Department of Computer Science, OsloMet – Oslo Metropolitan University, P.O. Box 4 St. Olavs plass, N-0130 Oslo, Norway*
[b] *Department of Computer Science, NTNU, Trondheim, Norway*
[c] *AI Lab – OsloMet Artificial Intelligence Lab, Pilestredet 52, N-0166 Oslo, Norway*
[d] *NordSTAR – Nordic Center for Sustainable and Trustworthy AI Research, Pilestredet 52, N-0166 Oslo, Norway*
[e] *Department of Holistic Systems, Simula Metropolitan Center for Digital Engineering, Pilestredet 52, N-0166 Oslo, Norway*
[f] *Department of Computer Science and Communication, Østfold University College, B R A Veien 4, N-1757 Halden, Norway*

ARTICLE INFO

Article history:
Received 23 February 2022
Received in revised form 23 August 2022
Accepted 24 August 2022
Available online 1 September 2022
Communicated by V.M. Pérez-Garcìa

Keywords:
Stochastic cellular automata
Criticality
Self-organization

ABSTRACT

There is evidence that biological systems, such as the brain, work at a critical regime robust to noise, and are therefore able to remain in it under perturbations. In this work, we address the question of robustness of critical systems to noise. In particular, we investigate the robustness of stochastic cellular automata (CAs) at criticality. A stochastic CA is one of the simplest stochastic models showing criticality. The transition state of stochastic CA is defined through a set of probabilities. We systematically perturb the probabilities of an optimal stochastic CA known to produce critical behavior, and we report that such a CA is able to remain in a critical regime up to a certain degree of noise. We present the results using error metrics of the resulting power-law fitting, such as Kolmogorov–Smirnov statistic and Kullback–Leibler divergence. We discuss the implication of our results in regards to future realization of brain-inspired artificial intelligence systems.

1. Introduction

Critical phenomena in general share two important features [1]. First, they show an infinite correlation length, which means that information spans throughout several scales, both in time and in space. Second, they occur between two well-defined phases each one assuming a specific range of values of an observable which can be "tuned" or controlled by one or more parameters. These parameters, called control parameters, can drive the system to switch between phases and for specific values, so-called critical values, one observes critical behavior. While most critical phenomena share these two ingredients, the second one is sometimes not observed, which means the system still shows correlation spanning through different scales, but there is no explicit control parameter to be tuned: independently of how we tune the system and initialize it, the system always evolves *towards* the critical state. In other words, the critical state is a stable state, an attractor of the system. This sort of critical behavior is called self-organized criticality (SOC), which is one of the most striking non-linear phenomena found in nature. Since its discovery in the 80s [2,3], several natural phenomena have been reported as showing SOC, ranging from the stock market [4–6] to the brain [7,8].

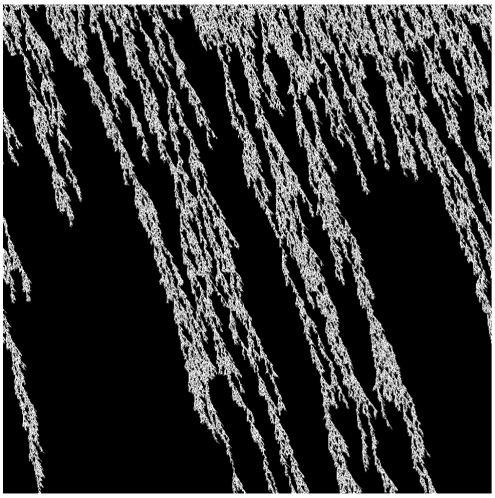
Indeed, brain functioning involves the coordination of neural activity across several scales, ranging from few neurons to large brain neural networks, leading to a natural resemblance to critical phenomena [9], and since this functioning is the "natural" state of the brain, it may be reasonable to hypothesize that this criticality follows some principles of self-organization. Recent discussions and investigations point indeed towards the possibility that SOC is the main factor for intelligence in the human brain [7] and therefore such findings from physics of critical phenomena may help to investigate how to learn from brain dynamics in order to improve the capacity of computation of artificial intelligence (AI) systems. How can self-organized criticality emerge in simple computational systems?

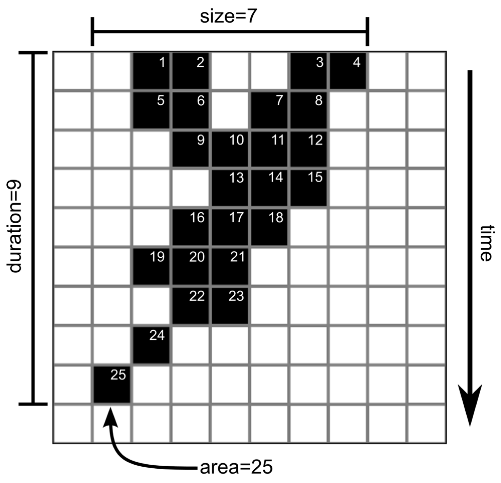
In this work, we address this question using cellular automata (CAs). CAs comprehend a family of models, in which a set of elementary "cells" form a lattice, typically with one or two dimensions [10]. The lattice iteratively evolves, and each cell takes one of a countable number of states. CAs are, therefore, models with discrete space, discrete time, and discrete state space.

Due to their simple implementation, they have been used to approach several complex phenomena, namely those showing large-scale correlations as a result of short-range, typically nearest neighbor, interactions [11]. The update of each cell considers

* Corresponding author at: Department of Computer Science, OsloMet – Oslo Metropolitan University, P.O. Box 4 St. Olavs plass, N-0130 Oslo, Norway.
E-mail address: sidneyp@oslomet.no (S. Pontes-Filho).

https://doi.org/10.1016/j.physd.2022.133507


(a) Optimal stochastic CA



(b) Avalanche definition

Fig. 1. (a) Illustration of the evolution of an optimal stochastic CA for critical behavior. The CA started with $N = 1000$ lattice squares and its evolution is plotted for $T = 1000$ time steps from top to bottom. The number of avalanches and the distributions of this illustration is shown in Fig. 2 (first row). (b) Illustration of the three properties characterizing an avalanche: its area, defined by the total number of active sites in the same connected set; its size, given by the total number of different cells (sites) belonging to the avalanche, at least at one iteration; its duration, given by the total number of (successive) iterations that include at least one site in the avalanche. In this example, we have size of 7 cells, a duration of 9 iterations, and a total area of 25 sites. Active cells are black and have state 1. The boundary of the CA also limits an avalanche.

the composition of its state together with the state of its nearest neighbors. From the very beginning, studies on CAs have also reported the emergence of critical behavior in general [12,13] and SOC in particular [14], where a prototypical example is the so-called Game of Life [15]. Recently, the concept of SOC in Game of Life was used in the context of machine behavior to introduce collective robots with simple control and local interactions [16], as well as for discussing the general features of artificial life [17].

The rules governing the evolution of each cell and its coupling with nearest neighbors can be deterministic: state configurations of one cell and its neighboring cells impose always the same state on the cell in the next iteration. A more realistic extension of such a model is to enable the iteration throughout CA's evolution to be updated according to some rules, with a certain *probability* p not necessarily 0 or 1. Such CAs are usually called stochastic CAs [18,19]. An example of a one-dimensional stochastic CA is shown in Fig. 1(a). Indeed, if the brain functioning shows critical behavior, it should involve critical states which can be achieved through dynamical processes driven by some stochastic freedom. Stochastic CAs seem therefore to be a better choice to explore criticality in the context of artificial intelligence.

Recently [18], using a genetic algorithm (please see [20] for an introduction to genetic algorithms), the authors found eight *optimal* probabilities that grant critical behavior in a 1D stochastic CA with 3 neighbors. Due to its freedom, this stochastic CA can have different outcomes, starting from the same initial configuration of the cells composing the CA, all of them showing critical behavior. One question which remains unanswered is how *sensitive* this stochastic CA is to small changes of the optimal probabilities. The brain functioning is driven by dynamical processes with some stochasticity, but it is also robust against changes in its stochastic features. Namely, its critical behavior is observed even under changes in the stochastic dynamics. The goal of this work is to assess the *robustness* of the critical behavior in the *optimal* stochastic CA presented in [18].

We start in Section 2 by introducing the main tools and methods, namely, how the *optimal* stochastic CA is obtained, the tools to uncover the critical behavior in the evolution of one particular CA and the measures to quantitatively assess the robustness of its critical behavior. In Section 3, we present our main results, reporting a broad range of probabilities for which criticality remains in the evolution of the CA. Section 4 concludes the paper.

2. Methods

2.1. Background and prior model

A CA that is one-dimensional, 2-state, and based on local interactions evolves according to an update rule of the form

$$c_{i,t+1} = \mathcal{F}(c_{i-1,t}, c_{i,t}, c_{i+1,t}), \tag{1}$$

where $c_{i,t}$ is either 0 or 1, denoting the state of cell i in iteration t. Periodic boundary conditions, $c_{0,t} = c_{N,t}$ and $c_{N+1,t} = c_{1,t}$, are used where N is the number of cells in the CA. In general, the function $\mathcal{F}$ maps each of the 8 possible 3-tuple at iteration t into the updated state of the middle cell i in the next iteration $t + 1$. If the CA is deterministic, there are exactly $2^8 = 256$ possible choices of $\mathcal{F}$.

If the CA is stochastic, instead of such $\mathcal{F}$-functions, we define a function $\mathcal{P}$ which gives the probability for the state at cell i in iteration $t + 1$ to be *one*, given the present state of the 3-tuple defining its nearest neighborhood:

$$\mathcal{P}(c_{i-1,t}, c_{i,t}, c_{i+1,t}) = \mathrm{Pro}\Big(\big[c_{i,t+1} = 1\big]\big|c_{i-1,t}, c_{i,t}, c_{i+1,t}\Big). \tag{2}$$

Function $\mathcal{P}$ is fully described by a vector of eight probabilities, one for each possible 3-tuple $(c_{i-1,t}, c_{i,t}, c_{i+1,t})$.

In Fig. 1(a), we present one example of a stochastic CA with periodic boundary condition, states uniformly initialized, and state

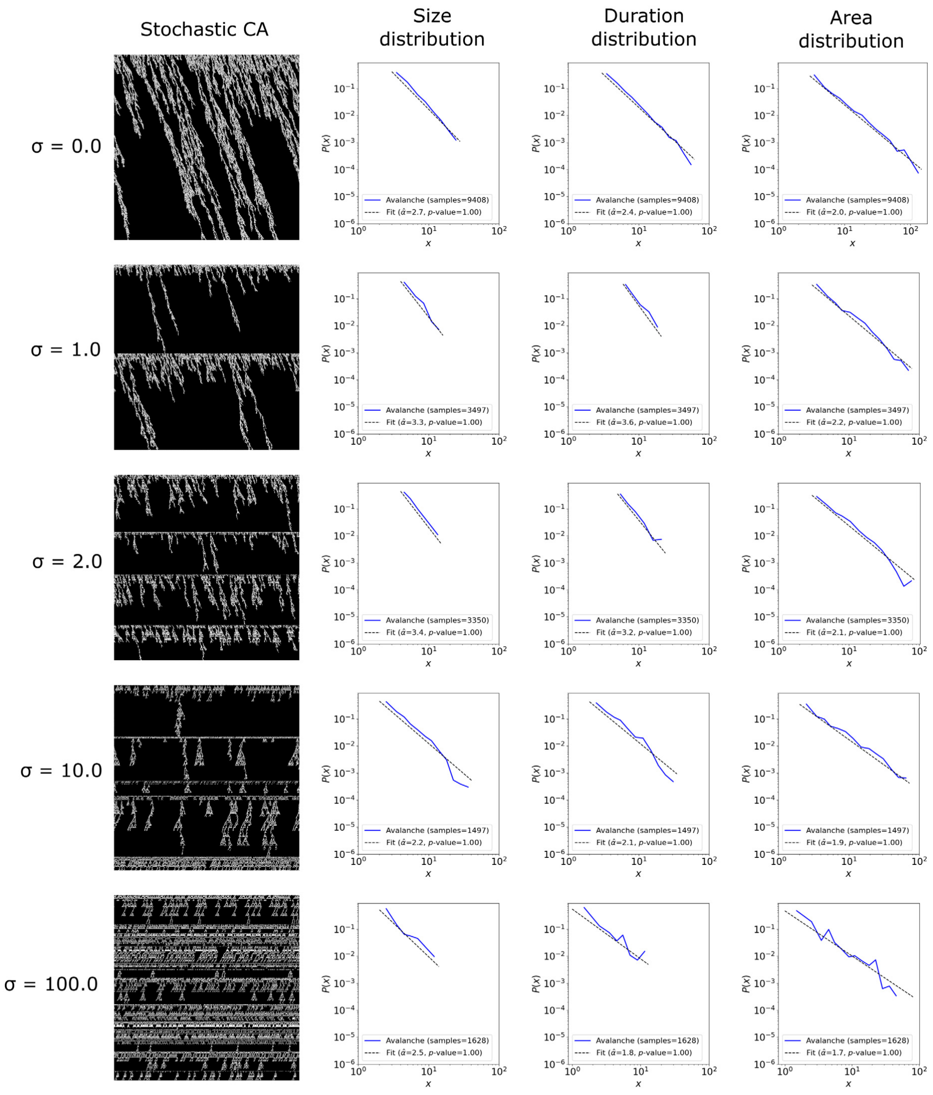


Fig. 2. Stochastic CAs and their avalanche distributions of the cells with state 1 (black cells) while affected by Gaussian noise. Here the CAs are produced with $N = 1000$ lattice squares and $T = 1000$ time steps. Large avalanches that happen only one time are ignored in these distributions. The dashed lines indicate the power-law fit, with a slope defined as $\hat{\alpha}$ and a goodness-of-fit given by the p-value (see text).

transition probabilities shown in Table 1. This choice of probabilities *maximizes* the critical behavior of the CA in such a configuration of boundary condition and initialization. There is a dependency in this configuration because if the CA starts with too many cells in the strong quiescent state (state 1), the resting state can be reached faster, then requiring re-initialization because all the cells have the same state and become static. With the re-initialization, the CA will maintain its activity and more avalanches can be produced. Also, some boundary conditions can make the CA never reach the resting state. The criticality of the CA will be assessed by measuring the statistical distribution of features characterizing an avalanche in the CA evolution. We define an avalanche during the evolution of a specific CA as the connected cluster of active states – i.e. state-1 cells – throughout the chain of cells composing the CA and throughout its time evolution. Fig. 1(b) illustrates an avalanche together with its three main features. The area of the avalanche corresponds to the total number of active states throughout the CA and during the full evolution which forms a connected set. The avalanche size is given by the total number of adjacent cells which are activated ($c = 1$) at least once, during the avalanche. The avalanche duration is given by the total number of successive iterations from the beginning until the end of the avalanche, including at least one site in the avalanche. The boundary of the CA is also

Table 1
Selected stochastic CA in [18].

State at t	$\mathcal{P}$ (Eq. (2))
(0,0,0)	0.394221
(0,0,1)	0.094721
(0,1,0)	0.239492
(0,1,1)	0.408455
(1,0,0)	0.000000
(1,0,1)	0.730203
(1,1,0)	0.915034
(1,1,1)	1.000000

the end of an avalanche. In the example of Fig. 1(b) the avalanche has a size of 7 cells, a duration of 9 iterations, and a total area of 25 sites. Our initial goal was to have a CA as wide and lasting as possible because we could get a broader range of avalanche size, duration and area. However, due to memory and time limitations, our selected number of lattice squares is $N = 1000$ and the number of time steps is $T = 1000$.

By keeping track of the avalanches emerging during the evolution of a stochastic CA, we can record the distribution of their area, size and duration. These three types of avalanche measurements are more similar to neuronal avalanches [7]. Moreover, the closer the distribution is from a power-law the closer the CA behavior is from a critical regime. Therefore, we recently introduced a fitness score (defined in Eqs. (4) and (5)), to assess how "critical" a particular CA realization is [18], based on different measures of the error associated with the fit. Using a genetic algorithm, we found the set of eight probabilities that leads to the most critical CA realizations (see Table 1).

Recently we implemented a Python library called *EvoDynamic*, to simulate several dynamical systems [18], such as cellular automata, random Boolean networks, and echo state networks. This library is based on the *TensorFlow* deep neural network framework [21], and it includes genetic algorithms that can be used to evolve dynamical systems towards a desired behavior or dynamics.

EvoDynamic has a general and single implementation that can simulate various dynamical systems because they are, in essence, dynamical graphs or networks. Since the weighted adjacency matrix and mapping function are specific to a dynamical system, they need to be adapted to simulate a stochastic elementary cellular automaton because the implementation of EvoDynamic is based on artificial neural networks. To make this possible, the implementation of an artificial neural network is generalized to incorporate the possibility of simulating CA. More specifically, EvoDynamic uses the general form of a feed-forward artificial neural network (ANN) without bias, $\mathbf{l}_{i+1} = a(\mathbf{W} \cdot \mathbf{l}_i)$, with $\mathbf{l}_i$, the layer index i of the ANN; $\mathbf{W}$, a weight matrix; and a, an activation function, and interprets it as a non-linear dynamical system with discrete time steps, namely:

$$\mathbf{c}_{t+1} = f(\mathbf{A} \cdot \mathbf{c}_t). \tag{3}$$

where $\mathbf{c}_t$ is a group of cells (or nodes in a graph) in iteration t, $\mathbf{A}$ is the weighted adjacency matrix, and f represents the (non-linear) mapping function. For the stochastic CA, the weighted adjacency matrix $\mathbf{A}$ connects the center cell $c_{i,t}$ with its three neighbors $(c_{i-1,t}, c_{i,t}, c_{i+1,t})$. To identify which neighbor has state 0 and 1 after the matrix multiplication between $\mathbf{A}$ and $\mathbf{c}_t$, the weights assigned to $(c_{i-1,t}, c_{i,t}, c_{i+1,t})$ are (4, 2, 1). Therefore, each of the eight neighborhood combinations is represented by a unique number from 0 to 7, which the mapping function f maps it to its corresponding probability for the random generation of state 0 or 1 in the next iteration.

The EvoDynamic framework was used to find the eight probabilities in Table 1 which maximizes the critical behavior of a stochastic CA, applying a so-called genetic algorithm. The genetic algorithm, in general, mimics a "natural selection" process, searching for optimal parameter values (the probabilities) by changing the parameter values and maximizing a pre-defined fitness function that calculates a fitness score. Instead of maximizing the function through a "supervised" path, such as a gradient descendent scheme, the genetic algorithm updates the values of the probabilities, starting from some set of initial values, and then applying random perturbations. If the perturbation increases the fitness score, the set of probabilities increases the chance to be selected as a "parent" for the next generation of new sets of probabilities. In the end, the genetic algorithm retrieved the "genotype" of the stochastic CA with the best fitness score, composed of the eight probabilities p_i ($i = 0, \ldots, 7$) for each of the eight 3-tuple $(c_{i-1,t}, c_{i,t}, c_{i+1,t})$.

The fitness function is based on the fitness measures of the avalanche size and duration with a power-law function and is heuristically defined as [18]

$$S_{temp} = (R^2)^2 + D^2 + B + U, \tag{4}$$

$$S = \begin{cases} S_{temp} + L, & S_{temp} > 3.5 \\ S_{temp}, & \text{otherwise.} \end{cases} \tag{5}$$

where R^2 is the coefficient of determination of complete linear fitting [22], D is the normalized coefficient of the Kolmogorov-Smirnov (KS) statistic [23], B is the percentage of non-zero bins with size one in the avalanche histograms, U is the percentage of unique states through time, and L is the normalized log-likelihood ratio of the comparison between the power-law model and the exponential model for estimating the avalanche distributions [23]. These fitness function objectives are normalized to the range [0, 1] if necessary and the genetic algorithm is applied to indirectly maximize them through the fitness function. The squared values in Eq. (4), R^2 and D, are the most important ones for the fitness function and were empirically chosen. In Eq. (5), the adjusted log-likelihood ratio L is only calculated if $S_{temp} > 3.5$ because this is a computationally intensive process; and if L is not trustworthy (p-value of the ratio is greater or equal to 0.1), then this measurement is ignored (set as zero). In the end, we obtain the fitness score S.

2.2. Adapted model with stochastic transition rates

To test the robustness of the stochastic CA for remaining in criticality, Gaussian noise with a varying standard deviation σ is applied to affect the probabilities p_i of the CA for every time step. Since the Gaussian noise will make the perturbed probabilities pass the valid range between zero and one, normalization to the Gaussian mean (original probability) is used. Thus, the equation for the normalization is

$$\tilde{\mu}_i = \log\left(\frac{p_i}{1 - p_i}\right). \tag{6}$$

With the normalized mean $\tilde{\mu}_i$, we sample x_i from a Gaussian distribution, such as

$$x_i \sim \mathcal{N}(\tilde{\mu}_i,\ \sigma). \tag{7}$$

The random variable x_i is sampled each time step, preserving the definition of CA, i.e. at each time step all cells follow the same (probabilistic) updating rule. To make x_i a valid perturbed probability $\tilde{p}_i$, the sigmoid function is applied to it, then

$$\tilde{p}_i = \frac{1}{1 + e^{-x_i}}. \tag{8}$$

While there are other possible choices, this choice maps a Gaussian distributed variable into a sigmoid function between 0 and 1 which for the average of x_i, $\tilde{\mu}_i$ retrieves the initial value of p_i.

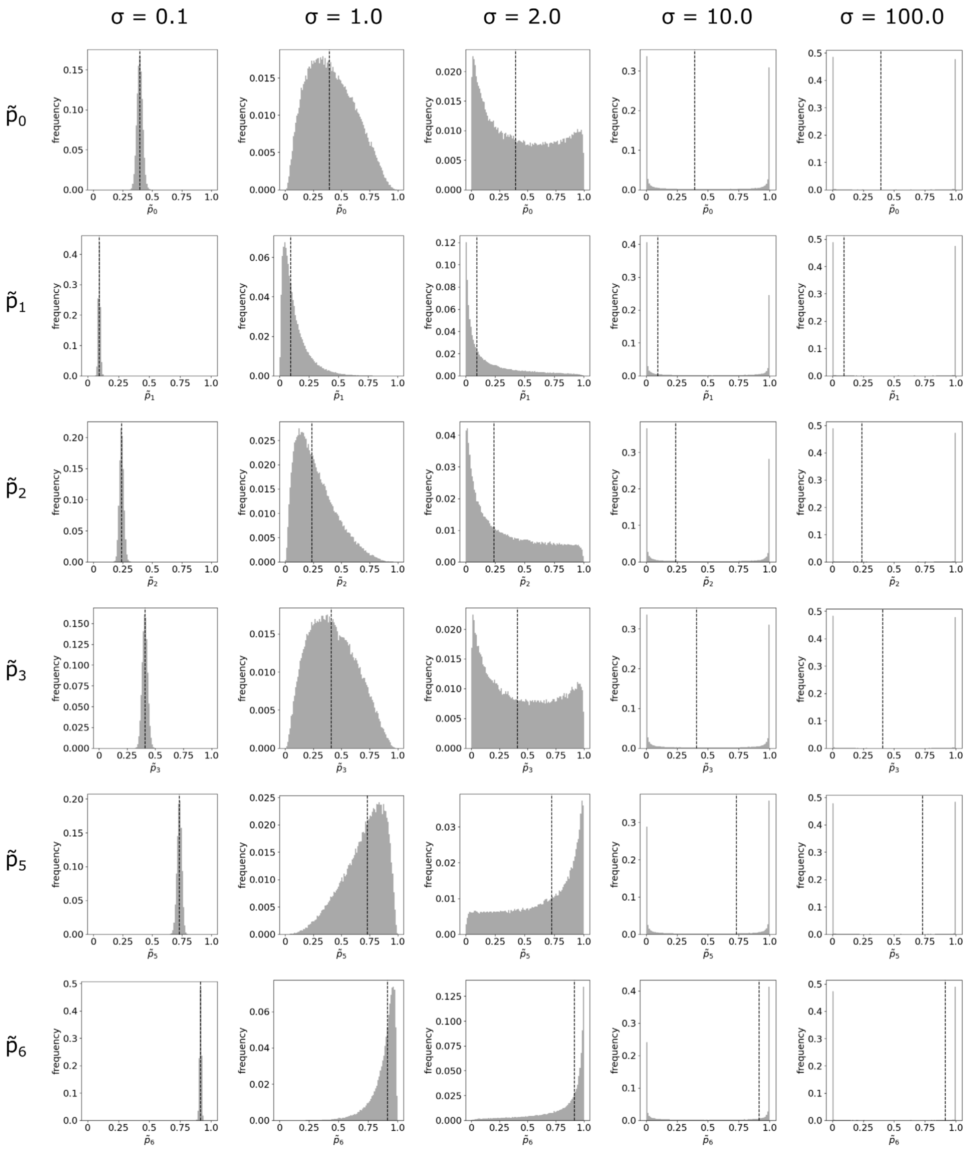
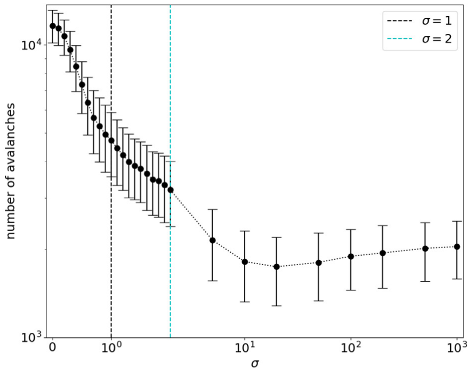


Fig. 3. Histogram of the valid perturbed probability $\tilde{p}_i$ for different values of σ. The vertical dashed line indicates the unperturbed probability ($\sigma = 0$). The probabilities $\tilde{p}_4$ and $\tilde{p}_7$ are not shown because they are always 0 and 1, respectively, independent of the value of σ. Each histogram was generated with 100,000 samples.

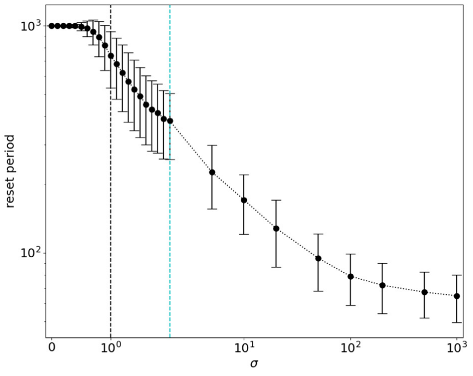
Having settled this, our research question can be reformulated as follows: for $\sigma = 0$ we have exactly the optimal solution, i.e. a power-law of the size of clusters, their duration, and their area, then, by increasing $\sigma > 0$ one will eventually destroy the power-laws observed for the optimal solution, pushing the system away from criticality. So, is there a transition from critical to non-critical stage tuning the σ? We note that the maximum value of σ for which criticality is still observed can be thought of as a measure of the robustness of the optimal (critical) stage.

3. Results

For producing the perturbed stochastic CAs and the distribution of their area, size and duration; we repeat the simulations for the values of the standard deviation σ from 0.1 to 2.0 in step size of 0.1, and also for values of 5, 10, 20, 50, 100, 200, 500, and 1000. For each of those values of σ, we perform 1000 CA simulations with a uniform random initialization. This is presented for some values of σ in Fig. 2. Its first row shows an illustrative realization



(a)



(b)

Fig. 4. Mean and standard deviation of (a) the number of avalanches and (b) reset period.

of the optimal stochastic CA with the probabilities in Table 1, showing the distribution of avalanche size, duration and area. A distribution is plotted as the avalanche measurement x with its occurrence probability $P(x)$. The power-law fit is indicated with dashed lines and we can see that a power-law distribution fits well the empirical histogram. The power-law fit is also indicated by its estimated slope $\hat{\alpha}$ and goodness-of-fit given by the p-value. This p-value is introduced by Clauset et al. [23], stating that a p-value greater than 0.1 indicates a valid power-law fit.

Fig. 3 presents the histograms of the valid perturbed probability $\tilde{p}_i$ calculated through Eq. (8) for some values of σ. When $\sigma = 2.0$, it is noticeable that the original Gaussian distribution becomes distorted. By increasing σ even more, the distribution of the valid perturbed probability $\tilde{p}_i$ becomes concentrated in 0 and 1.

Notice that, in case the strong quiescent state fully occupies the CA, its cells' states are re-initialized. For larger values of σ, since the CA reaches the inactivity state quicker than the optimal and unperturbed CA, the re-initialization happens more frequently. In Fig. 4, we plot the number of avalanches as well as the number of iterations before a re-initialization caused by all cells being in the quiescent state. As one sees in this figure, due to a more frequent re-initialization the average avalanche size, duration and area is reduced. In Fig. 2, this is not noticeable in the avalanche distributions because the large avalanches that happen just one time during the simulations are ignored. In those samples of the simulations, all goodness-of-fit p-value remains 1.0 as for the unperturbed CA. The standard deviation σ of the Gaussian noise to the probabilities strongly reduces the number of avalanches, especially from σ between 0.3 and 10.0. After that, the number of avalanches stabilizes around 2,000. This is because the avalanches became so short that they reach the resting state with all cells in the strong quiescent state much faster, then re-initializing the CA more often and having a similar number of avalanches. Such a reduction in the number of avalanches of state 1 is also due to the decrease in the occurrence of state 0 because the avalanches of state 1 need to be surrounded by cells with state 0 or boundary. We can perceive that the state 0 patterns became not only shorter but also thinner. To confirm this behavior and to have a sense of what happens at individual cell scale, Fig. 5 shows the occurrence rates and their standard deviation of state 0 and 1, and the transitions between these two states in both directions, from 0 to 1 ($0 \to 1$) and from 1 to 0 ($1 \to 0$). The occurrence rate of state 1 tends to increase until $\sigma = 2.0$

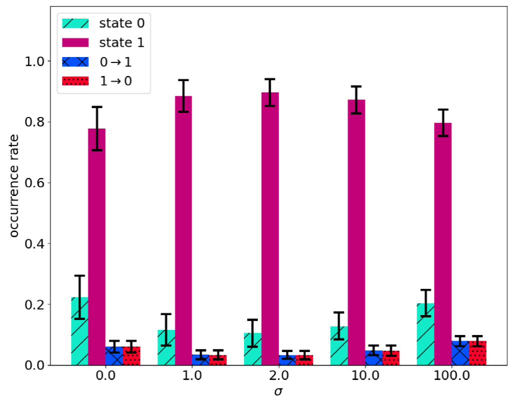


Fig. 5. Occurrence rates in the cells and their standard deviations of state 0 and 1, and the transitions between them that are from 0 to 1 ($0 \to 1$) and from 1 to 0 ($1 \to 0$). The statistics are calculated for each cell in 1000 CA simulations with $N = 1000$ and $T = 1000$.

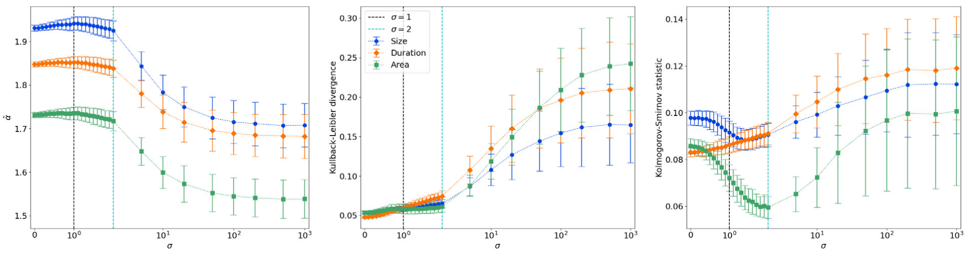
because it is the strong quiescent state. The occurrence rate of state 0 and the transitions are inversely or directly proportional to the rate of state 1 because they are interdependent. However, for $\sigma > 2.0$, the re-initialization of the CA counterbalances the trend of changes in those rates, even though it does not occur with the number of avalanches.

For the systematic evaluation of the robustness, we analyze the distributions of duration, size and area of the avalanches, by fitting to them a power-law and estimate their slope $\hat{\alpha}$, as well as their uncertainty through a Kolmogorov-Smirnov test and the corresponding Kullback-Leibler (KL) divergence [24,25].

The Kullback–Leibler divergence is defined as:

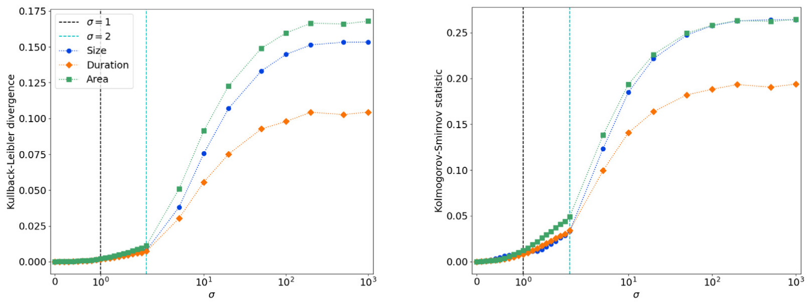
$$D_{KL} = \sum_x P(x) log \left(\frac{P(x)}{Q(x)} \right) \tag{9}$$

where $P(x)$ is the probability distribution of the empirical data and $Q(x)$ is the probability distribution function of the estimated



(a) Estimated slope $\hat{\alpha}$ (b) Kullback-Leibler divergence (c) Kolmogorov-Smirnov statistic

Fig. 6. Mean and standard deviation of the measurements of the power-law estimation for the avalanche distributions of state 1.



(a) Kullback-Leibler divergence (b) Kolmogorov-Smirnov statistic

Fig. 7. Comparison between the empirical distributions of each σ and the empirical distribution with $\sigma = 0$ used as a baseline.

power-law, and the supremum measuring the KS test is given by

$$D_{KS} = \sup_x |P_C(x) - Q_C(x)| \tag{10}$$

where $P_C(x)$ is the cumulative distribution of $P(x)$ and $Q_C(x)$ is the cumulative distribution function of $Q(x)$.

Results are shown in Fig. 6: the slope is plotted in Fig. 6(a), while the KL divergence is plotted in Fig. 6(b) and the KS test (supremum) statistic is shown in Fig. 6(c). Clearly, the slopes are almost constant for all distributions – size, duration and area – till a value of $\sigma \gtrsim 1$. Above $\sigma = 2$, the estimated slopes reduce drastically, indicating the deviation from critical behavior.

The KL divergence D_{KL} presents a constant (small) value for $\sigma \lesssim 1$, indicating a constant small error associated with the power-law fit. Beyond $\sigma = 2$ this error increases considerably. The KS statistic D_{KS} shows a more fluctuating behavior. Indeed, counter-intuitively, before reaching $\sigma = 1$, the supremum decreases to values smaller than the ones observed for $\sigma < 1$. This may be due to the occurrence of many small avalanches and one single large avalanche.

A comparison between the avalanche distributions of each σ and the avalanche distributions of the stochastic CA with $\sigma = 0$ is shown in Fig. 7. Therefore, avalanche distributions with $\sigma = 0$ are used as a baseline for KL divergence (Fig. 7(a)) and KS statistic (Fig. 7(b)). This comparison indicates how the dynamics changes when additional noise is applied, and it confirms our observations from the KL divergences with their power-law fits.

4. Discussion

The optimal stochastic CA without perturbations ($\sigma = 0$) and probabilities shown in Table 1 can maintain its critical behavior even when perturbed ($\sigma > 0$), but only up to a certain point. By analyzing Figs. 6 and 7, we can conclude that $\sigma = 1$ is the breaking point between behaving and not behaving similarly to the unperturbed one. This can be noticed especially in Fig. 6(a). The estimated slopes $\hat{\alpha}$ for size, duration and area remain almost unvaried until $\sigma = 1$, while their standard deviations slowly increase as a result of the fluctuations in the values of the probabilities. Fig. 4(a) shows that the number of avalanches starts to decrease from approximately 12,000 to around 4000 in the range $\sigma = [0, 1]$, even though the behavior is still maintained with respect to the estimated slopes $\hat{\alpha}$, KS statistic, and KL divergence. This cannot be seen in the reset period (Fig. 4(b)) because the number of time steps simulated is 1000, making it the maximum reset period possible.

Neuronal stochastic variability [26] happens all over the brain and on all scales. Since the optimal stochastic CA investigated in this work presented robustness up to perturbations with $\sigma = 1$, biological neural networks may also allow for similar robustness to perturbations. Therefore, both systems may preserve criticality even in noisy conditions. Because self-organized criticality is possibly one of the main factors for the emergence of intelligence in the human brain [7], the evaluation of robustness to stochastic variability can be an important measurement for indicating the presence of intelligence in artificial systems.

In future work, we plan to evaluate the performance of the optimal stochastic CA under perturbations in a machine learning framework called reservoir computing [27–29]. Reservoir computing is a biologically-plausible neural model inspired by the functioning of cortical microcircuits [30]. There is indeed evidence that reservoir computing achieves better performances when it produces critical dynamics [31–33]. This benchmark would inform how the artificial intelligence system can maintain its accuracy while increasing the probability noise. Therefore, such investigation may confirm that this robustness is also essential for future artificial intelligence systems, and in particular inform the realization of novel brain-inspired neuromorphic hardware.

CRediT authorship contribution statement

Sidney Pontes-Filho: Conceptualization, Methodology, Software, Visualization, Validation, Formal analysis, Data curation, Writing – original draft. **Pedro G. Lind:** Conceptualization, Methodology, Supervision, Writing – review & editing. **Stefano Nichele:** Supervision, Resources, Project administration, Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work was supported by the Norwegian Research Council, Norway SOCRATES project (grant number 270961).

References

[1] K. Christensen, Complexity and Criticality, Imperial College Press, 2005.
[2] P. Bak, C. Tang, K. Wiesenfeld, Self-organized criticality: An explanation of the 1/f noise, Phys. Rev. Lett. 59 (4) (1987) 381.
[3] P. Bak, C. Tang, K. Wiesenfeld, Self-organized criticality, Phys. Rev. A 38 (1) (1988) 364.
[4] B.B. Mandelbrodt, Fractals and Scaling in Finance, Springer-Verlag New York, 1997.
[5] J.P. da Cruz, P.G. Lind, The bounds of heavy-tailed return distributions in evolving complex networks, Phys. Lett. A 377 (2013) 189–194.
[6] J.P. da Cruz, P.G. Lind, The dynamics of financial stability in complex networks, Eur. Phys. J. B 85 (2012) 256.
[7] K. Heiney, O. Huse Ramstad, V. Fiskum, N. Christiansen, A. Sandvig, S. Nichele, I. Sandvig, Criticality, connectivity, and neural disorder: a multifaceted approach to neural computation, Front. Comput. Neurosci. 15 (2021) 7.
[8] A.J. Fontenele, N.A.-P. de Vasconcelos, T. Feliciano, L.A.-A. Aguiar, C. Soares-Cunha, B. Coimbra, L.D. Porta, S. Ribeiro, A.J. Rodrigues, N. Sousa, P.V. Carelli, M. Copelli, Criticality between cortical states, Phys. Rev. Lett. 122 (2019) 208101.
[9] L. Cocchi, L.L. Gollo, A. Zalesky, M. Breakspear, Criticality in the brain: A synthesis of neurobiology, models and cognition, Prog. Neurobiol. 158 (2017) 132–152.
[10] S. Wolfram, A New Kind of Science, Wolfram Media, 2002.
[11] G. Vichniac, Simulating physics with cellular automata, Physica D 10 (1984) 96–116.
[12] H. Chaté, P. Manneville, Criticality in cellular automata, Physica D 45 (1–3) (1990) 122–135.
[13] J. Singha, N. Gupte, Chimera states in coupled map lattices: Spatiotemporally intermittent behavior and an equivalent cellular automaton, Chaos 30 (11) (2020) 113102.
[14] P. Bak, K. Chen, M. Creutz, Self-organized criticality in the'Game of Life, Nature 342 (6251) (1989) 780–782.
[15] E.R. Berlekamp, J.H. Conway, R.K. Guy, Winning Ways for Your Mathematical Plays, Vol. 4, AK Peters/CRC Press, 2004.
[16] I. Rahwan, M. Cebrian, N. Obradovich, J. Bongard, J.-F. Bonnefon, C. Breazeal, J.W. Crandall, N.A. Christakis, I.D. Couzin, M.O. Jackson, et al., Machine behaviour, Nature 568 (7753) (2019) 477–486.
[17] C. Gershenson, V. Trianni, J. Werfel, H. Sayama, Self-organization and artificial life, Artif. Life 26 (3) (2020) 391–408.
[18] S. Pontes-Filho, P. Lind, A. Yazidi, J. Zhang, H. Hammer, G.B. Mello, I. Sandvig, G. Tufte, S. Nichele, A neuro-inspired general framework for the evolution of stochastic dynamical systems: Cellular automata, random boolean networks and echo state networks towards criticality, Cogn. Neurodyn. 14 (5) (2020) 657–674.
[19] C.G. Langton, Computation at the edge of chaos: Phase transitions and emergent computation, Physica D 42 (1–3) (1990) 12–37.
[20] J.H. Holland, Genetic algorithms, Sci. Am. 267 (1) (1992) 66–73.
[21] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D.G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, X. Zheng, TensorFlow: A system for large-scale machine learning, in: 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16), USENIX Association, Savannah, GA, 2016, pp. 265–283, URL: https://www.usenix.org/conference/osdi16/technical-sessions/presentation/abadi.
[22] S. Wright, Correlation and causation, J. Agric. Res. 20 (1921) 557–580.
[23] A. Clauset, C.R. Shalizi, M.E. Newman, Power-law distributions in empirical data, SIAM Rev. 51 (4) (2009) 661–703.
[24] S. Kullback, R.A. Leibler, On information and sufficiency, Ann. Math. Stat. 22 (1) (1951) 79–86.
[25] D.J. MacKay, D.J. Mac Kay, et al., Information Theory, Inference and Learning Algorithms, Cambridge University Press, 2003.
[26] M.D. McDonnell, J.H. Goldwyn, B. Lindner, Neuronal stochastic variability: influences on spiking dynamics and network activity, Front. Comput. Neurosci. 10 (2016) 38.
[27] M. Lukoševičius, H. Jaeger, Reservoir computing approaches to recurrent neural network training, Comp. Sci. Rev. 3 (3) (2009) 127–149.
[28] O. Yilmaz, Reservoir computing using cellular automata, 2014, arXiv preprint arXiv:1410.0162.
[29] S. Nichele, A. Molund, Deep reservoir computing using cellular automata, 2017, arXiv preprint arXiv:1703.02806.
[30] W. Maass, T. Natschläger, H. Markram, Real-time computing without stable states: A new framework for neural computation based on perturbations, Neural Comput. 14 (11) (2002) 2531–2560.
[31] L.E. Suárez, B.A. Richards, G. Lajoie, B. Misic, Learning function from structure in neuromorphic networks, Nat. Mach. Intell. 3 (9) (2021) 771–786.
[32] T.E. Glover, P. Lind, A. Yazidi, E. Osipov, S. Nichele, The dynamical landscape of reservoir computing with elementary cellular automata, in: ALIFE 2021: The 2021 Conference on Artificial Life, MIT Press, 2021.
[33] J. Boedecker, O. Obst, J.T. Lizier, N.M. Mayer, M. Asada, Information processing in echo state networks at the edge of chaos, Theory Biosci. 131 (3) (2012) 205–213.

Paper B.1

A conceptual bio-inspired framework for the evolution of artificial general intelligence
(Pontes-Filho and Nichele, 2019)

Author(s):

Sidney Pontes-Filho and Stefano Nichele

To appear at conference:

The 3rd Special Session on Biologically Inspired Parallel and Distributed Computing, Algorithms and Solutions (BICAS 2020) at The 18th International Conference on High Performance Computing and Simulation (HPCS 2020)

Copyright:

Note:

Due to the delay in the publication of the proceedings, it is presented its preprint in arXiv:1903.10410 [cs.NE,cs.AI]

Paper B.1
Pontes-Filho and
Nichele (2019)

A Conceptual Bio-Inspired Framework for the Evolution of Artificial General Intelligence

Sidney Pontes-Filho*,† and Stefano Nichele*,‡
*Department of Computer Science, Oslo Metropolitan University, Oslo, Norway
†Department of Computer Science, Norwegian University of Science and Technology, Trondheim, Norway
‡Department of Holistic Systems, Simula Metropolitan, Oslo, Norway
Email: sidneyp@oslomet.no, stenic@oslomet.no

***Abstract*—In this work, a conceptual bio-inspired parallel and distributed learning framework for the emergence of general intelligence is proposed, where agents evolve through environmental rewards and learn throughout their lifetime without supervision, i.e., self-learning through embodiment. The chosen control mechanism for agents is a biologically plausible neuron model based on spiking neural networks. Network topologies become more complex through evolution, i.e., the topology is not fixed, while the synaptic weights of the networks cannot be inherited, i.e., newborn brains are not trained and have no innate knowledge of the environment. What is subject to the evolutionary process is the network topology, the type of neurons, and the type of learning. This process ensures that controllers that are passed through the generations have the intrinsic ability to learn and adapt during their lifetime in mutable environments. We envision that the described approach may lead to the emergence of the simplest form of artificial general intelligence.**

***Index Terms*—Bio-inspired AI, Self-learning, Embodiment, Conceptual framework, Artificial General Intelligence**

I. Introduction

The brain is a truly remarkable computing machine that continuously adapts through sensory inputs. Rewards and penalties are encoded and learned throughout the evolution of organisms living in an environment (our world) that continuously provides unlabeled and mutable data. The supervision in the brain is a product of such evolutionary process. A real-world environment does not provide labeled data or predefined fitness functions for organisms and their brains as in supervised and reinforcement learning in Artificial Intelligence (AI) systems. However, organisms selected by natural and sexual selection [1] know or are able to learn which sensory inputs or input sequences may affect positively or negatively their survival and reproduction. One of the key components which ensures that a species will reproduce is the lifetime of the organisms. Pleasure, joy, and desire (or other positive inputs) may increase the lifetime of an organism and act as rewards. Pain, fear, and disgust may decrease the lifetime and act as penalties. All those feelings and emotions are results of the evolutionary pressure for increasing the life expectancy and succeeding in generating offspring [2]. One example is the desire and disgust that arise for some smells. The desire may come from the smell of nutritive food which increases life expectancy, while the disgust may come from spoiled food which may cause food poisoning, and therefore causing a lifetime reduction. Evolution by natural selection made it possible for living beings to be "interpreters" of sensory inputs by being attracted to rewards and repulsed by penalties, even though their first ancestors did not know what was beneficial or harmful in the surrounding environment.

Artificial General Intelligence (AGI) or strong AI has been pursued for many years by researchers in many fields, with the goal of reproducing human-level intelligence in machines, e.g., the ability of generalization and self-adaptation. So far, the AI scientific community has achieved outstanding results for specific tasks, i.e., weak or narrow AI. Such narrow AI implementations require highly specialized high performance computing systems for the training process. In this work, we propose to tackle the quest for general intelligence in its simplest form through evolution. It is therefore essential to develop a mutable environment that mimics what the first living beings with the simplest nervous systems faced. We define the simplest form of artificial general intelligence as the ability of an organism to continuously self-learn and adapt in a continuously changing environment of increasing complexity. Our definition of self-learning covers the meaning of self-supervised learning [3] and self-reinforcement learning [4]. Therefore, a self-learning agent is capable of interpreting the reactions of the surrounding environment affected by the agent's actions, then the sensory inputs of the agent are utilized to supervise or reinforce itself. This occurs because the sensory inputs contain cues for the agent to learn on its own. For example, the pain after touching a candle's flame (as the authors experienced in their childhood).

In this work, we propose the Neuroevolution of Artificial General Intelligence (NAGI) framework. NAGI is a bio-inspired framework which uses plausible models of biological neurons, i.e., spiking neurons [5], in an evolved network structure that controls a sensory-motor system in a mutable environment. Evolution affects the connection structure of neurons, their neurotransmitters (excitatory and inhibitory), and their local bio-inspired learning algorithms. The inclusion of such learning algorithms under evolutionary control is an important factor to generate long-term associative memory neural networks which may have cells with different plasticity rules [6]. Moreover, the genotype of the NAGI's agents does not contain the synaptic strength (weights) of the connections to avoid any innate knowledge about the environment. However, controllers that are selected for reproduction are those rewarded for their ability of self-learning and adaptation to new environments, i.e., an artificial newborn brain of an agent is an "empty box" with the innate ability to learn how to survive in different environments during its lifetime. That is somewhat similar to how humans have a specialized brain part for learning quickly any language when they are born [2].

The remainder of this paper is organized as follows. It provides background knowledge for the proposed NAGI framework, and then describes related works. Afterward, it contains a detailed explanation of the conceptual framework. Finally, we conclude this work by discussing the relevance of our approach for current AGI research, and we elaborate on possible future works which may include such a novel bio-inspired parallel and distributed learning method.

II. Background

The proposed NAGI framework brings together several key approaches in artificial intelligence, artificial life, and evolutionary robotics [7], briefly reviewed in this section. A Spiking Neural Network (SNN) is a type of artificial neural network that consists of biologically plausible neuron models [5]. Such neurons communicate with spikes or binary values in time series. SNNs incorporate the concept of time by intrinsically modeling the membrane potential within each neuron. Neurons spike when the membrane potential reaches a certain threshold. When the signals propagate as neurotransmitters to the neighboring neurons, their membrane potentials are therefore affected, increasing or decreasing. While SNNs are able to learn through unsupervised methods, i.e., Hebbian learning [8] and Spike-Timing-Dependent Plasticity (STDP) [9], spike trains are not differentiable and cannot be trained efficiently through gradient descent. NeuroEvolution of Augmenting Topologies (NEAT) [10] is a method that uses a Genetic Algorithm (GA) [11] to grow the topology of a simple neural network and adjust the weights of the connections to optimize a target fitness function, while allowing to keep diversity (speciation) in the population and to maintain compatible gene crossover with historical marking. The neuroplasticity used to adapt the weights in the proposed NAGI framework includes the Hebbian learning rule as STDP. In particular, the weight adaptation of STDP happens when the neuron produces a spike or action potential through the axon (i.e., output connection). Such an event allows the modification of the synaptic strength of the dendrites (i.e., input connections) that caused or did not cause that spike.

Funes and Pollack [12] describe the body/brain interaction (sensors and actuators vs. controller) as "chicken and egg" problem; the course of natural evolution shows a history of body, nervous system, and environment all evolving simultaneously in cooperation with, and in response to, each other [13]. Embodied evolution [14] is an evolutionary learning method for training agents through embodiment, i.e., embodied agents learning in an environment. Thus, in nature, general intelligence is a result of evolved self-learning through embodiment.

III. Related work

The idea of neuroevolution with adaptive synapses is not new. Stanley et al. [15] present NEAT with adaptive synapses using Hebbian local learning rules, with the goal of training neural networks for controlling agents in an environment. The authors verify the difference in performance with and without adaptation on the dangerous food foraging domain. The differences between their approach and ours are that their environment is static throughout the agent lifetime and they have inheritance of synaptic strength. Their results show that both networks with and without adaptive synapses reach the maximum fitness on that domain, and therefore both present "adaptation". An extended version of the previous method is Adaptive Hypercube-based NEAT (HyperNEAT) [16]. Adaptive HyperNEAT includes indirect encoding of the network topology as large geometric patterns.

A recent work that uses NEAT and no weight inheritance is Weight Agnostic Neural Networks (WANN), introduced by Gaier and Ha [17]. WANN is tested successfully with different supervision and reinforcement learning tasks whose weights are randomly initialized. Such promising results demonstrate that the network topology is as important as the connection weights in an artificial neural network. This is one of the motivations for the development of the NAGI framework. WANN and NEAT with adaptive synapses have been shown to be successful methods. However, such methods miss an important component for self-learning, which is a mutable environment as proposed in this work.

A recent review of neuroevolution can be found in [18] and it shows how competitive NEAT and its extensions are in comparison to deep neural networks trained with gradient-based methods for reinforcement learning tasks. Neuroevolution provides several extensions, which include indirect encoding to allow scalability, novelty search to promote diversity, meta-learning for training a network to learn how to learn, and the combination with deep learning for searching deep neural network architectures. Furthermore, its authors envisage that neuroevolution will be a key factor to reach AGI through meta-learning and open-ended evolution. However, in NEAT the neural weights are inherited, so there is no explicit target for general intelligence and adaptation.

A framework for the neuroevolution of SNNs and topology growth with genetic algorithms is proposed by Schaffer [19], with the goal of pattern generation and sequence detection. Eskandari et al. [20] propose a similar framework for artificial creature control, where the evolutionary process modifies and inherits the network topology and the SNN weights to perform a given task.

A method which tries to produce general intelligence incrementally is PathNet [21], where deep neural network paths are selected through evolution to perform the forward propagation and weight adjustment. Such evolving selection allows the network to learn new tasks faster by re-using frozen (previously learned) paths without catastrophic forgetting. Another framework that tries to produce low-level general intelligence is described by Voss [22]. It is a functional proof-of-concept prototype, owned by the company Adaptive A.I. Inc., which can interact with virtual and real world through sensors and actuators. Its controller, which has conceptual general intelligence capabilities, consists of a memory to save all data and to store the proprietary cognitive algorithms.

Multi-agent environments have also been considered a valuable stepping-stone towards AGI because the behavior of agents must adapt to cooperate and compete among them. One of the first examples of such multi-agent environment is the PolyWorld ecological simulator, introduced by Yaeger [23]. PolyWorld is a simulated environment of randomly generated food where evolving artificial organisms controlled by neural networks with Hebbian learning live. Organisms are able to eat, mate, fight, move, change the field of view, and utilize body brightness as a form of emergent communication. Their emergent behaviors are to some extent similar to the ones found in nature. Another recent multi-agent environment is presented by Lowe et al. [24]. However, in such an environment, the adaptation occurs in the actor-critic methods of their reinforcement learning framework. Such method outperforms traditional reinforcement learning approaches on competitive and cooperative multi-agent environments. Another reinforcement learning method which exploits multi-agent environments is introduced by Jaderberg et al. [25]. In their work, they use the environment of Quake III Arena Capture the Flag, a 3D first-person multiplayer game. Their method in this game exceeds human-level performance, therefore the artificial agents are able to cooperate and compete among them and even with human players. One work that provides an open-source competitive multi-agent environment for research purposes is Neural MMO [26]. Here, the agents are players which need to survive and prosper in an environment similar to the ones used in Massively Multiplayer Online Role-Playing Games (MMORPGs).

One important aspect of natural evolution is the ability to endlessly produce diverse solutions of increasing complexity, i.e., open-ended evolution (OOE). In contrast, OOE is difficult to achieve in artificial systems. A conceptual framework for the implementation of OOE in evolutionary systems is presented by Taylor [27]. Embodiment plays a key role in OOE in the context of the agent and its morphology, as discussed by Bongard [28]. For an articulated summary and discussion of OOE see [29]. In [30] the authors argue that open-ended evolution is a grand challenge for artificial general intelligence, and artificial life methods are promising starting points for the inclusion of OOE in AI systems.

IV. Framework concept

The main concept of the proposed NAGI framework is to mimic as close as possible the evolution of general intelligence in biological organisms, starting from the simplest form. To do that, we propose a minimalistic model with the following components. An agent is equipped with a randomly-initialized minimal spiking neural network. The agent is placed in a mutable environment in order to be able to generalize (learn to learn), instead of merely learning to solve the specific environment. Agents are more likely to survive if they perform correct actions. Agents have access to the environment through sensory inputs. The environment also provides intrinsic rewards and penalties. New agents inherit the topologies of the controllers from the previous generation (untrained), with the possibility of complexification (e.g., new neurons and synapses can appear through genetic operators). Training or learning happens throughout a generation. The goal of the untrained inherited controllers is to possess a topology that supports the ability to learn new environments. Neural learning occurs by utilizing environmental information (sensory input and environmental rewards/penalties) and neuroplasticity (e.g., Hebbian learning through spike-timing-dependent plasticity). The expected result is an unsupervised evolving system that learns without explicit training in a self-learning manner through embodiment. In this section, the components of the conceptual framework are described in details.

A. Data representation

The data that flows to and from the spiking neural networks that control the agents are encoded as firing rate, i.e., the number of spikes per second. The firing rate has minimum and maximum values, and is represented for simplification as real number between 0 and 1 (i.e., range $[0.0, 1.0]$). The stimulus to the neural networks can be Poisson-distributed spikes which have irregular interspike intervals, as observed in the human cortex [31]. That representation can be used for encoding input from binary environments (e.g., binary numbers 0 and 1 or Boolean values $False$ and $True$), or multi-value environments (e.g., represented as grayscale from black to white), and allows for representation of minimum and maximum activation values of sensors and actuators.

B. Self-Learning through Embodiment

A new agent learns through the reactions of an environment via embodiment (i.e., by having a "body" that affects an environment while sensing it). As such, the input of the neural network controller includes reward and penalty information for the learning process, such as the collision sensor in an autonomous robot whose activation represents a penalty, and a reward otherwise. This feedback information is the key factor for achieving self-learning. The concept of self-adaptation is closely connected with embodied cognition, a core property of living beings [32]. In contrast, supervised learning and reinforcement learning use the error of the neural network output to globally adjust the network model through methods of iterative error reduction, such as gradient descent. In embodied learning, the input itself is used to adjust the agent's controller. Such sensory input contains the reactions of the environment to the actions of the agent.

In the proposed framework, the local learning rules of the spiking neural network controller are responsible to correct the global behavior of the network according to agent experiences. This learning approach is, therefore, a result of self-learning through embodiment. The framework overview is depicted in Fig. 1. Note that self-learning through embodiment only works with agents in reactive environments (environments that affect the agents and are affected by them), such as any sensory-motor system deployed in the real-world. Non-reactive environments, on the other hand, do not react to any action of the agent, like any image classifier or object detector which only gives environmental information, thus there is no mutual interaction between an agent and a non-reactive environment. Therefore, we propose to create a virtual reactive environment for such cases. Virtual Embodied Learning (VEL) is the proposed method for such cases when no reward and penalty feedback is available through the sensory input. VEL adds reward and penalty inputs to a given sensory-motor system as illustrated in Fig. 1. It can also include the internal states of the agents, such as hunger and health. In addition, VEL can substitute supervised and reinforcement learning by using the loss of the model as penalty input and the opposite of the loss as reward input.

C. Mutable environment

To truly exploit and assess the self-learning capabilities and the generalization of the evolving spiking neural network, a mutable environment is proposed. The evolutionary goal of agents is to survive the changes in their environment. In the real-world, living organisms inherit modifications to their body and/or behavior through the generations. For example, a species may evolve a camouflage, such as the stick insects [33], and another one may evolve the appearance of a poisonous or venomous animal, such as the false coral snakes [34]. The proposed mutable environment is a simple metaphor of such examples.

Fig. 2a shows mutable environments that every agent in the population faces during its lifetime. Each agent has one sensor which provides one bit of information (i.e., $black$ or $white$) and can perform two actions (i.e., eat or $avoid$). In each generation, the agents are presented with environmental data from several environments. Each sample is presented for a given period to allow the agents' controllers to learn. In the first environment, the correct action eat is associated with the $white$ color while the action $avoid$ is associated with $black$. Once the environmental data has been consumed by the agent, there is an abrupt change in the interpretation of the environment ($black$ and $white$ are flipped) and the agents are presented with the environmental data again. Agents that perform well in many environments within each generation are more likely to go through the next generation.

Fig. 2b presents more complex mutable environments where agents have two sensors and non-binary environmental values can be received. As shown in the figure, different environments are procedurally generated and presented in each generation, where abrupt changes in the labeling of correct and wrong actions have happened. The set of actions may also be expanded to more than two, with different effects on agents' lifetime and their fitness scores.

D. Neuroplasticity

Each neuron in the evolved spiking neural network may have a different plasticity rule. The different types of learning rules are subject to evolutionary control. Examples of learning rules include asymmetric Hebbian, symmetric Hebbian, asymmetric anti-Hebbian, symmetric anti-Hebbian [9]. Together with all the Hebbian learning rules encoded in the genome, there will be the effectiveness of the potentiation and the depression of the synapse strengths, i.e., how strong the learning rules are going to be for reducing or increasing the weight of the synapses. Moreover, other types of learning rules discovered in neuroscience may be added together or in parallel to those, such as non-Hebbian learning, neuromodulation, and synapse fatigue [35]–[37]. The neuroplasticity will also be regulated by a maximum total value of synaptic strength that a neuron can have for its dendrites. In case this value is reached, the increase in the weight of a synapse will cause a decrease of the others in the same neuron. This type of weight normalization is reported in [38], [39] for biological neurons.

E. Neuroevolution

The population of genomes (spiking neural network controllers) for the agents is evolved through a modification of NEAT [10]. The genotypes of NEAT describe the topology and weights of the synapses, while our proposed method does not evolve the weights while includes in the genotype the type of neurotransmitter and neuroplasticity [9]. The weights of the spiking neural networks are randomly initialized in every generation because the agents should not have innate knowledge of the environment [2]. Therefore, the proposed framework focuses on the self-learning capabilities of the agents. Their lifetime will be longer when agents perform correct actions and shorter when they perform wrong actions. The lifetime of agents is used as fitness score to define the best performing neural networks.

Algorithm 1 explains how an agent's genome is evaluated while it is in a mutable environment during its lifetime. The fitness score for the agent is equal to the time the agent is alive until its death. Each agent has a maximum life expectancy. Such life expectancy is reduced faster when an agent receives a penalty and it is reduced slower when the agent receives a reward. Both penalties and rewards reduce the lifetime of agents, as one agent is not to live for an infinite amount of time if it always performs the correct action.

The neuroevolution process allows the growth of neural network topologies and therefore the population is initialized with minimal networks that complexify over time. Nevertheless, there may be a penalty on lifetime to avoid the generation of big networks which may have neuron groups that specialize for each different environment. Therefore it allows the network to learn how to forget the previous environment, and then be able to adapt to the new one [40]. Another reason to apply this penalty for the size of the network is that more neurons require more energy to maintain them. This reduction of lifetime caused by the number of neurons can be regulated

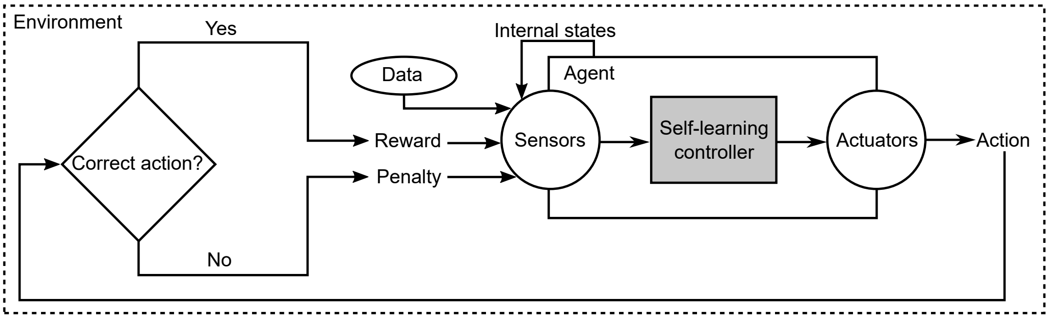


Fig. 1. Illustration of virtual embodied learning or self-learning through embodiment in a non-reactive environment. In the case of a reactive environment, rewards and penalties are embedded within the environmental data.

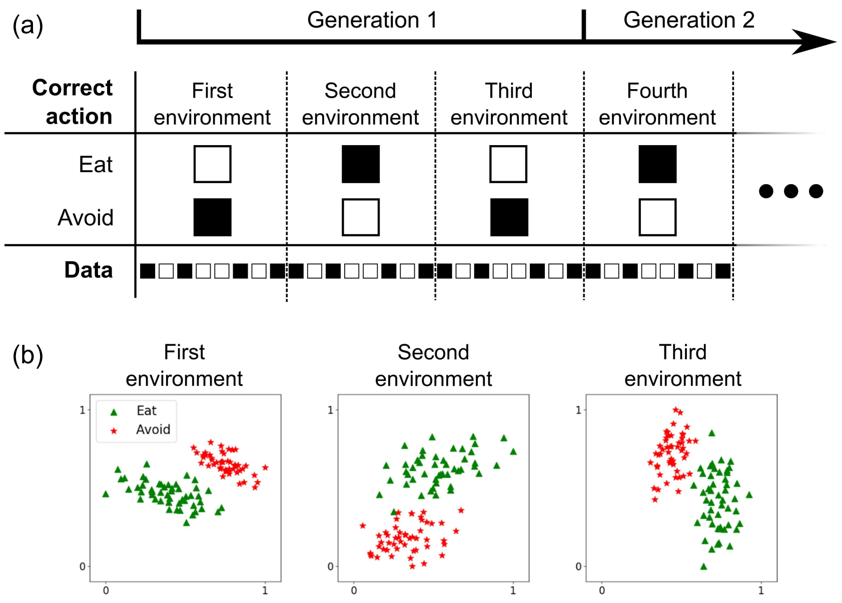


Fig. 2. Samples of mutable environments that can be presented to the agents through their evolution. Agents can execute two actions (eat or avoid). Within each generation, after the agents of a generation have seen all samples of an environment, a new one is presented. (a) 1D environment where the agent has one binary sensor. (b) The agent has two non-binary sensors (i.e, the axes).

by a parameter, therefore choosing it is of high importance to the fitness and lifetime of the agent.

V. Discussion and Conclusion

While current AI methods such as deep learning and reinforcement learning (and their combinations) have proven to be successful in solving a multitude of challenging tasks, e.g., defeating humans in the real-time strategy game Starcraft II [41], there is a lot of debate around the limitation of current methods for breakthroughs in Artificial General Intelligence. One key difference between AI and AGI is the learning ability. Most of AI methods (supervised, unsupervised, and reinforcement learning) are explicitly trained, while AGI needs some intrinsic ability to self-learn.

One of the open questions for AGI research is: how can artificial agents be able to acquire the general skill of learning, in order to continuously adapt throughout their lifetime?

In biological systems, we infer that self-learning is a result of rewards and penalties which are embedded in the sensory data living beings receive from the environment (unlabeled

Algorithm 1 Agent's genome evaluation using mutable environment and virtual embodied learning

```
 1: procedure EVALUATE($genome$)
 2:     $agent \leftarrow$ new Agent($genome$)        ▷ Agent is initialized with untrained neural network
 3:     $lifetime \leftarrow 0$
 4:     while $agent$ is alive do
 5:         if $dataset$ is empty then
 6:             $dataset \leftarrow getNextDatasetForCurrentGeneration()$        ▷ Next temporary environment of the current generation
 7:         $data, label \leftarrow getRandomSampleAndDeleteFrom(dataset)$
 8:         $reward \leftarrow False$        ▷ Initialization of $reward$
 9:         $penalty \leftarrow False$        ▷ Initialization of $penalty$
10:         while ($agent$ is learningSample) $\wedge$ ($agent$ is alive) do        ▷ Agent learns the presented sample with neuroplasticity for a period of time
11:             $lifetime \leftarrow lifetime + 1$
12:             $action \leftarrow agent(data, reward, penalty)$
13:             $reward \leftarrow action = label$
14:             $penalty \leftarrow action \neq label$
15:             $agent.healthReduction(reward, penalty)$        ▷ Penalty reduces the agent's health faster than reward, then accelerating its death
16:     return $lifetime$
```

and mutable data). Their ability to learn through this form of self-learning through embodiment is a result of evolution.

One of the goals of the proposed NAGI framework is an AGI system that allows the adaptation and general learning skills through the three main levels of self-organization in living systems [42]:

- Phylogeny, which includes evolution of genetic representations;
- Ontogeny, which takes care of the morphogenetic process (growth) from a single cell to a multicellular machine, by following the genotype instructions;
- Epigenesis, which allows the emergence of a learning system through an indirect encoding between genotype and phenotype, and the phenotype is subject to modifications (learning) throughout the lifetime while interacting with the environment.

We, therefore, envision the proposed spiking neural network model will include developmental and morphogenetic processes [43] in future extensions of the framework.

Another envisioned stepping-stone to AGI is the extension of the framework to artificial life multi-agent systems. Multi-agent environments will allow the emergence of more advanced strategies of adaptation and learning based on collaboration and competition. In addition, the framework may benefit from extending the environment itself into an evolving agent, which can also allow for increased complexity and open-ended evolution.

Finally, we expect that future implementations of the NAGI framework and its extensions will be deployed/embodied into real robot agents equipped with physical sensors.

In conclusion, this work proposes a novel general framework for the neuroevolution of artificial general intelligence (NAGI) in its simplest form, which can be extended to more complex tasks and environments. In NAGI, the general intelligence, i.e., learning to learn to adapt to different environments, is a result of self-learning through embodiment. Therefore, the learning process is not a result of explicit training with supervision or reinforcement learning, as there is no loss function used to adjust the neural network weights. The proposed neural network model is a bio-inspired model based on spiking neural networks. Their learning is based on spike-timing-dependent plasticity which uses only input data for learning. As such, penalties and rewards are embedded within the environmental data sensed by the agents.

This work describes the details of the NAGI conceptual framework as a novel paradigm for self-learning. Therefore, the experimental results are not included in this contribution. However, our current experimental results are promising, and part of a separate contribution.

The NAGI conceptual framework proposes a computational system which may allow the simplest form of general intelligence observed in nature to emerge. Self-learning through embodiment shifts the way machines currently learn by changing the paradigm of supervised and reinforcement learning. Our efforts are also to reduce the gap between biological neural networks computation and artificial intelligence implementations, allowing for a biologically-inspired neural network model that suits the paradigm in artificial life of massively parallel, distributed, and local interactions.

Acknowledgements

We thank Gustavo Moreno e Mello, Kristine Heiney, Anis Yazidi, and Benedikt Vogler for thoughtful comments and discussions. This work was supported by Norwegian Research Council SOCRATES project (grant number 270961).

References

[1] S. J. Arnold and M. J. Wade, "On the measurement of natural and sexual selection: theory," *Evolution*, vol. 38, no. 4, pp. 709–719, 1984.

[2] A. Zador, "A critique of pure learning: What artificial neural networks can learn from animal brains," *bioRxiv preprint bioRxiv:582643*, 2019.

[3] P. Sermanet, C. Lynch, Y. Chebotar, J. Hsu, E. Jang, S. Schaal, S. Levine, and G. Brain, "Time-contrastive networks: Self-supervised learning from video," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 1134–1141.

[4] B. Hilleli and R. El-Yaniv, "Toward deep reinforcement learning without a simulator: An autonomous steering example," in *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[5] E. M. Izhikevich, "Simple model of spiking neurons," *IEEE Transactions on neural networks*, vol. 14, no. 6, pp. 1569–1572, 2003.

[6] B. F. Grewe, J. Gründemann, L. J. Kitch, J. A. Lecoq, J. G. Parker, J. D. Marshall, M. C. Larkin, P. E. Jercog, F. Grenier, J. Z. Li *et al.*, "Neural ensemble dynamics underlying a long-term associative memory," *Nature*, vol. 543, no. 7647, p. 670, 2017.

[7] S. Doncieux, N. Bredeche, J.-B. Mouret, and A. E. G. Eiben, "Evolutionary robotics: what, why, and where to," *Frontiers in Robotics and AI*, vol. 2, p. 4, 2015.

[8] D. O. Hebb, *The organization of behavior: A neuropsychological theory*. New York: Wiley, Jun. 1949.

[9] Y. Li, Y. Zhong, J. Zhang, L. Xu, Q. Wang, H. Sun, H. Tong, X. Cheng, and X. Miao, "Activity-dependent synaptic plasticity of a chalcogenide electronic synapse for neuromorphic systems," *Scientific reports*, vol. 4, p. 4906, 2014.

[10] K. O. Stanley and R. Miikkulainen, "Evolving neural networks through augmenting topologies," *Evolutionary computation*, vol. 10, no. 2, pp. 99–127, 2002.

[11] J. H. Holland, "Genetic algorithms," *Scientific american*, vol. 267, no. 1, pp. 66–73, 1992.

[12] P. Funes and J. Pollack, "Evolutionary body building: Adaptive physical designs for robots," *Artificial Life*, vol. 4, no. 4, pp. 337–357, 1998.

[13] C. Mautner and R. K. Belew, "Evolving robot morphology and control," *Artificial Life and Robotics*, vol. 4, no. 3, pp. 130–136, 2000.

[14] R. A. Watson, S. G. Ficiei, and J. B. Pollack, "Embodied evolution: embodying an evolutionary algorithm in a population of robots," in *Proceedings of the 1999 Congress on Evolutionary Computation-CEC99 (Cat. No. 99TH8406)*, vol. 1, July 1999, pp. 335–342 Vol. 1.

[15] K. O. Stanley, B. D. Bryant, and R. Miikkulainen, "Evolving adaptive neural networks with and without adaptive synapses," in *The 2003 Congress on Evolutionary Computation, 2003. CEC'03.*, vol. 4. IEEE, 2003, pp. 2557–2564.

[16] S. Risi and K. O. Stanley, "Indirectly encoding neural plasticity as a pattern of local rules," in *International Conference on Simulation of Adaptive Behavior*. Springer, 2010, pp. 533–543.

[17] A. Gaier and D. Ha, "Weight agnostic neural networks," in *Advances in Neural Information Processing Systems*, 2019, pp. 5365–5379.

[18] K. O. Stanley, J. Clune, J. Lehman, and R. Miikkulainen, "Designing neural networks through neuroevolution," *Nature Machine Intelligence*, vol. 1, no. 1, pp. 24–35, 2019. [Online]. Available: https://doi.org/10.1038/s42256-018-0006-z

[19] J. D. Schaffer, "Evolving spiking neural networks: A novel growth algorithm corrects the teacher," in *2015 IEEE Symposium on Computational Intelligence for Security and Defense Applications (CISDA)*, May 2015, pp. 1–8.

[20] E. Eskandari, A. Ahmadi, S. Gomar, M. Ahmadi, and M. Saif, "Evolving spiking neural networks of artificial creatures using genetic algorithm," in *Neural Networks (IJCNN), 2016 International Joint Conference on*. IEEE, 2016, pp. 411–418.

[21] C. Fernando, D. Banarse, C. Blundell, Y. Zwols, D. Ha, A. A. Rusu, A. Pritzel, and D. Wierstra, "Pathnet: Evolution channels gradient descent in super neural networks," *arXiv preprint arXiv:1701.08734*, 2017.

[22] P. Voss, "Essentials of general intelligence: The direct path to artificial general intelligence," in *Artificial general intelligence*. Springer, 2007, pp. 131–157.

[23] L. Yaeger, "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, 1994, pp. 263–263.

[24] R. Lowe, Y. Wu, A. Tamar, J. Harb, O. P. Abbeel, and I. Mordatch, "Multi-agent actor-critic for mixed cooperative-competitive environments," in *Advances in Neural Information Processing Systems*, 2017, pp. 6379–6390.

[25] M. Jaderberg, W. M. Czarnecki, I. Dunning, L. Marris, G. Lever, A. G. Castaneda, C. Beattie, N. C. Rabinowitz, A. S. Morcos, A. Ruderman *et al.*, "Human-level performance in first-person multiplayer games with population-based deep reinforcement learning," *arXiv preprint arXiv:1807.01281*, 2018.

[26] J. Suarez, Y. Du, P. Isola, and I. Mordatch, "Neural mmo: A massively multiagent game environment for training and evaluating intelligent agents," *arXiv preprint arXiv:1903.00784*, 2019.

[27] T. Taylor, "Evolutionary innovations and where to find them: Routes to open-ended evolution in natural and artificial systems," *Artificial Life*, vol. 25, no. forthcoming, 2019.

[28] J. Bongard, N. Cheney, Z. Mahoor, and J. Powers, "The role of embodiment in open-ended evolution," *OOE3: The Third Workshop on Open-Ended Evolution*, 2018.

[29] T. Taylor, M. Bedau, A. Channon, D. Ackley, W. Banzhaf, G. Beslon, E. Dolson, T. Froese, S. Hickinbotham, T. Ikegami *et al.*, "Open-ended evolution: perspectives from the oee workshop in york," *Artificial life*, vol. 22, no. 3, pp. 408–423, 2016.

[30] K. O. Stanley, J. Lehman, and L. Soros, "Open-endedness: The last grand challenge youve never heard of," *O'Reilly*, 2017.

[31] D. Heeger, "Poisson model of spike generation," *Handout, University of Standford*, vol. 5, pp. 1–13, 2000.

[32] L. Smith and M. Gasser, "The development of embodied cognition: Six lessons from babies," *Artificial life*, vol. 11, no. 1-2, pp. 13–29, 2005.

[33] S. Lev-Yadun, A. Dafni, M. A. Flaishman, M. Inbar, I. Izhaki, G. Katzir, and G. Ne'eman, "Plant coloration undermines herbivorous insect camouflage," *BioEssays*, vol. 26, no. 10, pp. 1126–1130, 2004.

[34] T. M. Davidson and J. Eisner, "United states coral snakes," *Wilderness & Environmental Medicine*, vol. 7, no. 1, pp. 38–45, 1996.

[35] H. K. Kato, A. M. Watabe, and T. Manabe, "Non-hebbian synaptic plasticity induced by repetitive postsynaptic action potentials," *Journal of Neuroscience*, vol. 29, no. 36, pp. 11 153–11 160, 2009.

[36] J. P. Johansen, L. Diaz-Mataix, H. Hamanaka, T. Ozawa, E. Ycu, J. Koivumaa, A. Kumar, M. Hou, K. Deisseroth, E. S. Boyden *et al.*, "Hebbian and neuromodulatory mechanisms interact to trigger associative memory formation," *Proceedings of the National Academy of Sciences*, vol. 111, no. 51, pp. E5584–E5592, 2014.

[37] T. Abrahamsson, B. Gustafsson, and E. Hanse, "Synaptic fatigue at the naive perforant path–dentate granule cell synapse in the rat," *The Journal of physiology*, vol. 569, no. 3, pp. 737–750, 2005.

[38] S. Royer and D. Paré, "Conservation of total synaptic weight through balanced synaptic depression and potentiation," *Nature*, vol. 422, no. 6931, p. 518, 2003.

[39] S. El-Boustani, J. P. K. Ip, V. Breton-Provencher, H. Okuno, H. Bito, and M. Sur, "Locally coordinated synaptic plasticity shapes cell-wide plasticity of visual cortex neurons in vivo," *bioRxiv preprint bioRxiv:249706*, 2018.

[40] A. S. Benjamin, *Successful remembering and successful forgetting: A festschrift in honor of Robert A. Bjork*. Psychology Press, 2011.

[41] O. Vinyals, I. Babuschkin, J. Chung, M. Mathieu, M. Jaderberg, W. M. Czarnecki, A. Dudzik, A. Huang, P. Georgiev, R. Powell, T. Ewalds, D. Horgan, M. Kroiss, I. Danihelka, J. Agapiou, J. Oh, V. Dalibard, D. Choi, L. Sifre, Y. Sulsky, S. Vezhnevets, J. Molloy, T. Cai, D. Budden, T. Paine, C. Gulcehre, Z. Wang, T. Pfaff, T. Pohlen, Y. Wu, D. Yogatama, J. Cohen, K. McKinney, O. Smith, T. Schaul, T. Lillicrap, C. Apps, K. Kavukcuoglu, D. Hassabis, and D. Silver, "AlphaStar: Mastering the Real-Time Strategy Game StarCraft II," https://deepmind.com/blog/alphastar-mastering-real-time-strategy-game-starcraft-ii/, 2019.

[42] M. Sipper, E. Sanchez, D. Mange, M. Tomassini, A. Pérez-Uribe, and A. Stauffer, "A phylogenetic, ontogenetic, and epigenetic view of bio-inspired hardware systems," *IEEE Transactions on Evolutionary Computation*, vol. 1, no. 1, pp. 83–97, 1997.

[43] R. Doursat, H. Sayama, and O. Michel, *Morphogenetic engineering: toward programmable complex systems*. Springer, 2012.

Paper B.2

Towards the neuroevolution of low-level artificial general intelligence
(Pontes-Filho et al., 2022)

Author(s):
Sidney Pontes-Filho, Kristoffer Olsen, Anis Yazidi, Michael A. Riegler, Pål Halvorsen and Stefano Nichele

Published in journal:
Frontiers in Robotics and AI, vol. 9, 2022

Paper B.2
Pontes-Filho et al. (2022)

frontiers | Frontiers in Robotics and AI

TYPE Original Research
PUBLISHED 14 October 2022
DOI 10.3389/frobt.2022.1007547

Check for updates

OPEN ACCESS

EDITED BY
Sheri Marina Markose,
University of Essex, United Kingdom

REVIEWED BY
Stefania Costantini,
University of L'Aquila, Italy
Neil Vaughan,
University of Exeter, United Kingdom

*CORRESPONDENCE
Sidney Pontes-Filho,
sidneyp@oslomet.no

SPECIALTY SECTION
This article was submitted to Computational Intelligence in Robotics, a section of the journal Frontiers in Robotics and AI

RECEIVED 30 July 2022
ACCEPTED 03 October 2022
PUBLISHED 14 October 2022

CITATION
Pontes-Filho S, Olsen K, Yazidi A, Riegler MA, Halvorsen P and Nichele S (2022), Towards the Neuroevolution of Low-level artificial general intelligence. *Front. Robot. AI* 9:1007547.
doi: 10.3389/frobt.2022.1007547

Towards the Neuroevolution of Low-level artificial general intelligence

Sidney Pontes-Filho[1,2]*, Kristoffer Olsen[3], Anis Yazidi[1,4,5], Michael A. Riegler[6,7], Pål Halvorsen[1,6] and Stefano Nichele[1,4,5,6,8]

[1]Department of Computer Science, Oslo Metropolitan University, Oslo, Norway, [2]Department of Computer Science, Norwegian University of Science and Technology, Trondheim, Norway, [3]Department of Informatics, University of Oslo, Oslo, Norway, [4]AI Lab—OsloMet Artificial Intelligence Lab, Oslo, Norway, [5]NordSTAR—Nordic Center for Sustainable and Trustworthy AI Research, Oslo, Norway, [6]Department of Holistic Systems, Simula Metropolitan Centre for Digital Engineering, Oslo, Norway, [7]Department of Computer Science, UiT the Arctic University of Norway, Tromsø, Norway, [8]Department of Computer Science and Communication, Østfold University College, Halden, Norway

In this work, we argue that the search for Artificial General Intelligence should start from a much lower level than human-level intelligence. The circumstances of intelligent behavior in nature resulted from an organism interacting with its surrounding environment, which could change over time and exert pressure on the organism to allow for learning of new behaviors or environment models. Our hypothesis is that learning occurs through interpreting sensory feedback when an agent acts in an environment. For that to happen, a body and a reactive environment are needed. We evaluate a method to evolve a biologically-inspired artificial neural network that learns from environment reactions named Neuroevolution of Artificial General Intelligence, a framework for low-level artificial general intelligence. This method allows the evolutionary complexification of a randomly-initialized spiking neural network with adaptive synapses, which controls agents instantiated in mutable environments. Such a configuration allows us to benchmark the adaptivity and generality of the controllers. The chosen tasks in the mutable environments are food foraging, emulation of logic gates, and cart-pole balancing. The three tasks are successfully solved with rather small network topologies and therefore it opens up the possibility of experimenting with more complex tasks and scenarios where curriculum learning is beneficial.

KEYWORDS
neuroevolution, artificial general intelligence, spiking neural network, spike-timing-dependent plasticity, Hebbian learning, weight agnostic neural network, meta-learning

1 Introduction

Artificial General Intelligence (AGI) or strong Artificial Intelligence (AI) is commonly discussed among AI researchers. It is often defined as human-level AI. However, the generality of an AI does not need to be considered at such a level of complexity. Even an artificial neural network that performs lots of different tasks as a collection of specialized or weak AI (Reed et al., 2022) may not provide the level of generality observed in simple biological systems. In fact, our current artificial intelligent systems cannot emulate the adaptability to unknown conditions and learning capabilities of an animal with a simple nervous system, such as a worm (Ardiel and Rankin, 2010; Randi and Leifer, 2020). An alternative approach is to start the quest for the generality of AI from the simplest tasks that animals can do, but machines cannot, like behaving intelligently even in new environments (Crosby et al., 2019), i.e., out-of-distribution generalization (Shen et al., 2021). Moreover, AGI systems should be tested in tasks that require self-learning on the fly from sensory feedback, as it is often done in meta-learning and continual learning (Najarro and Risi, 2020; Zohora et al., 2021).

We argue that a radical paradigm change is needed in order to reach general intelligence (Lake et al., 2017; Crosby et al., 2019). Our hypothesis is that such a new paradigm requires learning systems with self-organizing properties, as discussed by Risi, (2021). In this work, our goal is to achieve the learning capabilities of a primitive brain. Therefore, we aim at a low-level AGI, i.e., a system that can learn a map function through sensory experience. Interpreting and understanding sensory inputs are achieved through evolution, particularly supervised evolution (Zador, 2019) of agents interacting with their environment.

The brain is the organ that interprets the encoded signals from our sensory organs, thanks to the ability to distinguish between positive and negative sensory experiences depending on what is considered to be good or harmful, e.g., pleasure and pain. The experiences of pleasure and pain serve as reward and penalty mechanisms that may affect our behavior by conditioning associative positive and negative cues with specific memories.

In this work, we evaluate the Neuroevolution of Artificial General Intelligence (NAGI) framework (Pontes-Filho and Nichele, 2019). NAGI is a low-level biologically-inspired AGI framework. NAGI consists of an evolvable spiking neural network with adaptive synapses and randomly-initialized weights. The network is evolved by an extension of the method NeuroEvolution of Augmenting Topologies (NEAT) (Stanley and Miikkulainen, 2002). The source code of NAGI is available at https://github.com/SocratesNFR/neat-nagi-python.

The evolved spiking neural network controls an agent placed in a mutable environment. Its chances of reproduction are proportional to how long it can survive in an environment that is constantly changing, sometimes abruptly. Evolution optimizes how the neurons are connected in the network, their type of neurotransmitters (excitatory or inhibitory), their susceptibility to background electrical current noise (analogous to bias), and their neuroplasticity. With such degrees of freedom in the optimization process, we attempt to approximately recapitulate the evolutionary process of the simplest brains. The mutable environment and random weight initialization propitiate a benchmark for generality and adaptivity of the agent.

We test NAGI in three mutable environments. The first one is a simple food foraging task, in which the agent has one photoreceptor (or light intensity sensor) used to identify food. The food type (color) is either black or white. Food can be edible or poisonous and this feature changes over time. The agent can also taste the food as its sensory feedback for good and bad actions. The second environment is a logic gate task. The spiking neural network needs to emulate different logic gates in series where the only reward and penalty sensory signals are the supporting mechanisms to identify the correct output. The third environment is a cart-pole balancing task. In this environment, the goal of the agent is to control the forces applied to the cart in order to maintain the pole above itself upright. The mutable component of this environment is the pole length, which changes during the lifetime of the agent. Because this environment has sensory feedback for the agent's actions, there is no need to add reward and penalty sensory signals.

The article is organized as follows: Section 2 explains the theoretical basis for understanding NAGI. Section 3 discusses the related work to our approach. Section 4 describes the details of the method and experiments. Section 5 presents the experimental results. Section 6 concludes the article including a discussion of the results and plans for future work.

2 Background

The components of the NAGI framework are inspired by the overlapping research fields of artificial life (Langton, 2019), evolutionary robotics (Doncieux et al., 2015), and computational neuroscience (Trappenberg, 2009). In particular, the controller for the agents is a Spiking Neural Network (SNN) (Izhikevich, 2003), which is a more biologically-plausible artificial neural network. The neurons in an SNN communicate through spikes, i.e., binary values in time series. Therefore, an SNN adds a temporal dimension to binary data. A neuron propagates such data depending on whether its membrane potential crossed a threshold value or not. If the threshold is crossed, the neuron propagates a signal represented as neurotransmitters to its connected neurons; otherwise, the action potential is not propagated. When neurotransmitters are released by a neuron, they can be of two types: excitatory, which increases the membrane potential and the likelihood of producing an action potential; or inhibitory, which has the opposite effect by decreasing the membrane potential. Efficient optimization of an SNN cannot happen through gradient descent as spike trains are not differentiable (Tavanaei et al., 2019). Instead, spiking neurons have biologically inspired local learning rules, such as Hebbian learning and Spike-Timing-Dependent Plasticity (STDP) (Hebb, 1949; Li et al., 2014). Those neuroplasticity rules are unsupervised, and their functionality in the brain is still not fully understood. However, it is inferred that the supervision comes from a certain network configuration acquired through evolution. Therefore, in this work, we use a modification of NeuroEvolution of Augmenting Topologies (NEAT) (Stanley and Miikkulainen, 2002). NEAT uses a Genetic Algorithm (GA) (Holland, 1992) to optimize the weights and the topology of a growing neural network that is initialized with a minimal and functional size. NEAT is typically used to search for a network configuration that improves a fitness score while maintaining population diversity (speciation) and avoiding loss of genes during crossover (historical marking). For an accessible and extensive explanation of NEAT, please refer to Ref. (Welleck, 2019).

A distinction from NEAT is that the weights in the NAGI framework are randomly initialized, and they change (adapt) after deployment. The adaptation is coordinated by a realistic Hebbian learning rule, i.e., STDP. This neuroplasticity adjusts the synaptic strength of a neuron's dendrites (i.e., input connections) when it fires an action potential (or spike) that goes through its axon (i.e., output connection). The weights are modified according to the difference in time between incoming spikes and the generated action potential. More detailed information about SNN and STDP is available in Ref. (Camuñas-Mesa et al., 2019).

The body and brain interaction (sensors and actuators vs. controller) is often described as "chicken and egg" problem (Funes and Pollack, 1998). The natural evolution of body and brain happens together with the evolution of the environment. They evolve in cooperation and response to each other (Mautner and Belew, 2000). The application of supervised evolution of agents interacting with the environment is defined as embodied evolution (Watson et al., 1999). As such, an agent needs a body to learn from the reaction of its environment. We hypothesize that low-level general intelligence in nature emerged through the evolution of a sensory feedback learning method.

3 Related work

Neuroevolution with adaptive synapses was introduced in 2003 by Stanley et al. (2003). Such a method is a version of NEAT where the synaptic strength of the connections changes with Hebbian local learning rules. In their work, they used a food foraging task where an agent moves around a field surrounded by edible and poisonous food. The type of food did not change over time, but it was initialized differently at every new run. The agents needed to try the food first before identifying it. Therefore, the agents possess reward and penalty sensory signals as in NAGI. This method is rather similar to ours. However, NAGI is more biologically plausible, weight agnostic, and is tested in a mutable environment. Risi and Stanley, (2010) proposed an extended version by replacing the direct encoding of the network in NEAT with an indirect encoding.

Additional related methods are described in Refs. (Gaier and Ha, 2019) and (Najarro and Risi, 2020) where randomly-initialized artificial neural networks are used. The work of Gaier and Ha, (2019) uses a version of NEAT where each neuron can have one activation function out of several types. While in the method of Najarro and Risi, (2020), the network topology is fixed and each connection evolves to optimize the parameters of its Hebbian learning rule.

In a recent review on neuroevolution (Stanley et al., 2019), NEAT and its extensions are comparable to deep neural networks trained with gradient-based methods for reinforcement learning tasks. Such methods allow evolving artificial neural networks with indirect encoding for scalability, novelty search for diversity, meta-learning for learning how to learn, and architecture search for deep learning models. Moreover, neuroevolution is described as a key factor for reaching AGI, particularly in relation to meta-learning and open-ended evolution. Meta-learning encompasses the training of a model with certain datasets and testing it with others. The goal of the model is therefore to learn any given dataset by itself from experience (Thrun and Pratt, 1998). Open-ended evolution is the ability to endlessly generate a variety of solutions of increasing complexity (Taylor, 2019). In NAGI, meta-learning is an implicit target in the mutable environments and is implemented as neuroplasticity in the spiking neural network.

In 2020, Nadji-Tehrani and Eslami, (2020) introduced the framework for evolutionary artificial general intelligence (FEAGI). This method uses an indirect encoding technique for a spiking neural network that resembles the growth of the biological brain, which is called "neuroembryogenesis." As a proof of concept, FEAGI demonstrates successful handwritten digits classification by learning through association and being able to recall digits from different image samples in real-time.

4 Neuroevolution of Artificial General Intelligence

The NAGI framework aims at providing a simplified model of the initial stages of the evolution of biological general intelligence (Pontes-Filho and Nichele, 2019). The evolving agents in NAGI consist of randomly-initialized spiking neural networks. Thus, a genome in NAGI does not require the definition of synaptic weights of the connections between neurons, as it is done in NEAT. Therefore, the synaptic weights in the genome are replaced by an STDP rule and its parameters for each neuron. Since biological neurons may provide one of the two main neurotransmitters, NAGI's genome defines such a feature in the neurons' genes. As such, a neuron can be either excitatory or inhibitory. To imitate the function of bias in artificial neural networks, neurons may be also susceptible to a "background electrical current noise."

The environment changes during the lifetime of the agent. This forces the agent to learn new environmental conditions. Therefore, the agent is encouraged to generalize and learn how to learn. The aforementioned random initialization and mutable environment aim at benchmarking the basic properties needed for low-level AGI.

4.1 Spiking neural network

The spiking neural network has a fixed number of input and output neurons depending on the task to be solved. The neuroevolution process defines the number of hidden neurons that will be available. Hidden neurons can be either excitatory or inhibitory, while input and output neurons are always excitatory. Self-loops and cycles are permitted while duplicate connections between two neurons in the same direction are prohibited. The SNN is stimulated from the input neurons, as such units are spike generators. The spikes are uniformly generated in an assigned frequency or firing rate.

As a spiking neuron model, we use a simplification of the leaky integrate-and-fire model (Liu and Wang, 2001). A neuron's membrane potential v is increased directly by its inputs and decays over time by a factor λ_{decay}. We can then express the change in membrane potential Δv with regards to a time step Δt by

$$\Delta v(\Delta t) = \sum_{i=1}^{n} w_i x_i - \Delta t \lambda_{decay} v, \tag{1}$$

where x_i is the input value 0 (no spike) or 1 (spike) from the presynaptic neuron i, the dendrite for this connection has the synaptic strength defined as w_i, and n is the total number of presynaptic neurons that the dendrites are connecting. If the membrane potential v is greater than the membrane threshold v_{th}, a spike is released and the membrane potential returns to the resting membrane potential v_{rest}, which is 0. The time step Δt we use in the experiments is 0.1 ms, and decay factor λ_{decay} is $0.01\Delta t$. An action performed by the SNN is calculated by the number of spikes in a time window. Such an actuator time window covers 250 ms or 2,500 time steps. In NAGI, the weights of the SNN are randomly initialized with a normal distribution. The mean is equal to 1 and the standard deviation is equal to 0.2. The weights are always positive. As mentioned, the excitation and inhibition of a neuron are defined by the neurotransmitter of the presynaptic neuron.

4.1.1 Homeostasis

Biological neurons have a plasticity mechanism that maintains a steady equilibrium of the firing rate, which is called homeostasis (Betts et al., 2013; Kulik et al., 2019). In our method, the spiking neurons can have non-homogeneous inputs, which could lead to very different firing rates. It is desirable that all neurons have approximately equal firing rates (Diehl and Cook, 2015). In order to homogenize the firing rates of the neurons in a network, the membrane threshold v_{th} is given by

$$v_{th} = \min\left(v_{th}^{*} + \Theta, \sum_{i=1}^{n} w_i\right), \tag{2}$$

where v_{th}^{*} is the "resting" membrane threshold equals to 1; and Θ starts with value 0, increases 0.2 every time a neuron fires, and decays exponentially with a rate of $0.01\Delta t$. Each neuron has an individual Θ. Therefore, a neuron firing more often will get a larger membrane threshold and consequently a lower firing rate. To compensate for a neuron with weak incoming weights, which causes a low firing rate; we instead use the sum of the incoming weights as the threshold.

4.1.2 Spike-Timing-Dependent Plasticity

The adjustment of the weights of the connections entering into a neuron happens on every input and output spike to and from a neuron. This is performed by STDP. It is done by keeping track of the time elapsed since the last output spike and each input spike from incoming connections within a time frame. Such a time frame is called the STDP time window and is set to be ±40 ms. The difference between presynaptic and postsynaptic spikes, or the relative timing between them, denoted by Δt_r is given by

$$\Delta t_r(t_{out}, t_{in}) = t_{out} - t_{in}, \tag{3}$$

where t_{out} is the timing of the output spike and t_{in} is the timing of the input spike.

The synaptic weight change Δw is calculated in accordance with one of the four Hebbian learning rules. The functions for each of the four learning rules are given by

$$\Delta w(\Delta t_r) = \begin{cases} A_+ e^{\frac{-\Delta t_r}{\tau_+}} & \Delta t_r > 0, \\ -A_- e^{\frac{\Delta t_r}{\tau_-}} & \Delta t_r < 0, \quad \text{Asymmetric Hebbian} \\ 0 & \Delta t_r = 0; \end{cases} \tag{4}$$

$$\Delta w(\Delta t_r) = \begin{cases} -A_+ e^{\frac{-\Delta t_r}{\tau_+}} & \Delta t_r > 0, \\ A_- e^{\frac{\Delta t_r}{\tau_-}} & \Delta t_r < 0, \quad \text{Asymmetric Anti-Hebbian} \\ 0 & \Delta t_r = 0; \end{cases} \tag{5}$$

$$\Delta w(\Delta t_r) = \begin{cases} A_+ g(\Delta t_r) & g(\Delta t_r) > 0, \\ A_- g(\Delta t_r) & g(\Delta t_r) < 0, \quad \text{Symmetric Hebbian} \\ 0 & g(\Delta t_r) = 0; \end{cases} \tag{6}$$

$$\Delta w(\Delta t_r) = \begin{cases} -A_+ g(\Delta t_r) & g(\Delta t_r) > 0, \\ -A_- g(\Delta t_r) & g(\Delta t_r) < 0, \quad \text{Symmetric Anti-Hebbian} \\ 0 & g(\Delta t_r) = 0; \end{cases} \tag{7}$$

where $g(\Delta t_r)$ is a Difference of Gaussian function given by

$$g(\Delta t_r) = \frac{1}{\sigma_+\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{\Delta t_r}{\sigma_+}\right)^2} - \frac{1}{\sigma_-\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{\Delta t_r}{\sigma_-}\right)^2}, \tag{8}$$

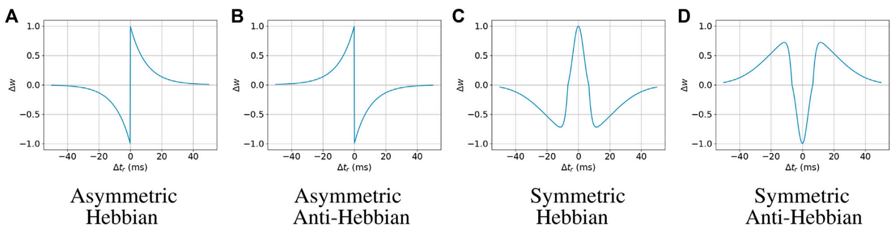


FIGURE 1
Spike-timing-dependent plasticity rules.

A_+ and A_- are the parameters that affect the height of the curve, τ_+ and τ_+ are the parameters that affect the width or steepness of the curve of the Asymmetric Hebbian functions, and σ_+ and σ_- are the standard deviations for the Gaussian functions used in the Symmetric Hebbian functions. It is also required that $\sigma_- > \sigma_+$. We experimentally found fitting ranges for each of these parameters, which are $A_+ = [0.1, 1.0]$, $A_- = [0.1, 1.0]$, $\tau_+ = [1.0, 10.0]$, and $\tau_- = [1.0, 10.0]$ for the asymmetric STDP functions; and $A_+ = [1.0, 10.6]$, $A_- = [1.0, 44.0]$, $\sigma_+ = [3.5, 10.0]$, and $\sigma_- = [13.5, 20.0]$ for the symmetric ones. The STDP curves with the maximum value of those parameters are illustrated in Figure 1.

Weights can take values in a range $[w_{\min}, w_{\max}]$, and every neuron has a weight budget w_{budget} it must follow. What this means is that if the sum of a neuron's incoming weights exceed w_{budget} after initialization or STDP has been applied, they are normalized to w_{budget}, given by

$$\text{if } \sum_{i=1}^{n} w_i > w_{budget}, \text{then } w_i = \frac{w_i w_{budget}}{\sum_{i=1}^{n} w_i}. \qquad (9)$$

The parameters used during our experiments are $w_{\min} = 0$, $w_{\max} = 1$, and $w_{budget} = 5$. In case of a SNN without homeostasis, if a connection i has $w_i = w_{\max}$, then $w_i = v_{th}$. Therefore, an action potential coming from i will always produce a spike. This is the reason why $w_{\max} = v_{th}$.

4.2 Genome

The genome in NAGI is rather similar to the one in NEAT. Its node genes have three types: input, hidden, and output. Depending on the type of the node gene, there is a different collection of *loci*[1]. The input node is a spike generator and provides excitation to the neurons it is connected to. The gene of an input node is the same as in NEAT. The hidden and output nodes represent adaptable and mutable spiking neurons. They have three additional loci: the type of the learning rule, the set of the learning rule parameters, and a bias. The connection gene in NAGI has no weight locus as in NEAT. The reason for its removal is that the weights of the SNN are defined by a normal distribution.

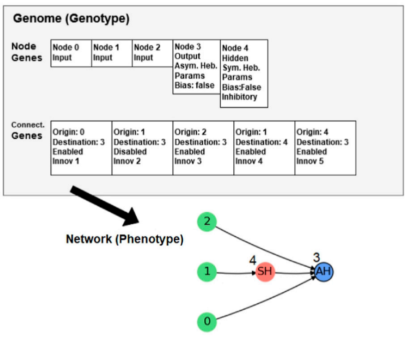

Genome (Genotype)
Node Genes
Node 0 Input
Node 1 Input
Node 2 Input
Node 3 Output Asym. Heb. Params Bias: false
Node 4 Hidden Sym. Heb. Params Bias:False Inhibitory
Connect. Genes
Origin: 0 Destination: 3 Enabled Innov 1
Origin: 1 Destination: 3 Disabled Innov 2
Origin: 2 Destination: 3 Enabled Innov 3
Origin: 1 Destination: 4 Enabled Innov 4
Origin: 4 Destination: 3 Enabled Innov 5
Network (Phenotype)
2
1
0
4
SH
3
AH

FIGURE 2
Genotype and phenotype in NAGI. Image taken from Ref. (Olsen, 2020).

The learning rule is one of the four STDPs. The set of learning rule parameters consists of four parameters that adjust the intensity of the weight change. They are different for symmetric and asymmetric learning rules. The symmetric parameters are $\{A_+, A_-, \sigma_+, \sigma_-\}$ and the asymmetric parameters are $\{A_+, A_-, \tau_+, \tau_-\}$. The bias is a Boolean value that determines if the neuron has a constant input of 0.001 being added to Δv, which is analogous to the background noise of the neuron.

1 In the terminology of genetic algorithms, a value within a gene is also called a *locus* (plural *loci*).

The hidden node genes have a unique locus, which is a Boolean value that determines whether it represents an inhibitory or excitatory neuron. This locus is not included in the output node genes because they are always excitatory. As a result of combining all the descriptions of the genome in NAGI, the genotype and the phenotype are illustrated in Figure 2.

The initialization of the additional loci in the node genes can be conditional and non-uniform. The initialization of the neurotransmitter type of a neuron follows a similar proportion of excitatory and inhibitory neurons in the brain (Sukenik et al., 2021). The probability of a neuron being added as excitatory is 70%. The probability of having a bias is 20%. Depending on the neurotransmitter, excitatory neurons have a 70% chance of initializing with Hebbian plasticity, and inhibitory neurons have the same chance but for anti-Hebbian plasticity. The learning rule parameters are initialized by sampling from a uniform distribution within the STDP parameter ranges.

The mutations of the additional loci happen in 10% of chance to switch the neurotransmitter type, bias, learning rule, and learning rule parameters. Those parameters have 2% chance of a fully re-initialization. When the parameters are assigned to be mutated, a random value sampled from a normal distribution with $\mu = 0$ and $\sigma^2 = m(p)$ is added to the parameter p. The equation of $m(p)$ is

$$m(p) = 0.2\left(p_{max} - p_{min}\right), \tag{10}$$

where $p_{\max}$ and $p_{\min}$ are the maximum and minimum values the parameter can have, given by the STDP parameter ranges. During the neuroevolution, 10% of the genotypes with the best fitness scores will be passed to the next generation unchanged, i.e., elitism.

4.3 Mutable environments

The benchmark tasks for NAGI are meant to evaluate the agent's ability to generalize and self-adapt. Therefore, they consist of environments that change during the lifetime of the agent. Two types of tasks are provided, binary classification (two tasks of this kind are provided) and control (one task of this kind is provided). The first type (binary classification) is the simplest one, however, it provides the most abrupt changes in the environment. The binary classification tasks are food foraging with one input, and logic gates with two inputs. The control task in a simulated physical environment is the cart-pole balancing from OpenAI Gym (Brockman et al., 2016). The changes are less abrupt in this last task as they consist in modifying the pole size. The fitness scores are calculated using the number of time steps t that the agent survived in these environments, normalized to the range [0, 1] using the maximum possible lifetime $L_{\max}$ and minimum possible lifetime $L_{\min}$. Therefore, the fitness function f is given by

$$f(t) = \frac{t - L_{min}}{L_{max} - L_{min}}. \tag{11}$$

In the binary classification tasks, the agents have an initial amount of health points that is reduced every time step as continuous damage. If a correct action is chosen, the health point amount is reduced by d_c health point. Otherwise, it is reduced by d_i. The input sample is given to the agent for 1 s or 10,000 time steps, then it is changed to a new one. The mutation of the environment condition happens when the agent has seen four samples. The order of the input samples and the environment conditions is fixed and cyclic.

We noticed that the number of spikes within the actuator time window can be the same for the output neurons and therefore allowing for a tie in many cases. Our solution to avoid spiking neural networks with this behavior is to include a "confidence" factor in the fitness score calculation. Therefore, the higher the difference between the spike count, the more confident the action is. If the action is correct and highly confident, the damage is d_c or closer. If the action is incorrect but highly confident, the damage is d_i or closer. The lack of confidence would make the damage lie between the values d_c and d_i. The spike count for the correct action s_c and incorrect one s_i are used to calculate the participation of the spikes for deciding the correct action p_c and the participation for the incorrect action p_i. In the iterations without spikes of the output neurons, normally the initial ones; the agent takes d_i as damage. Otherwise, the damage is calculated by

$$p_c(s_c, s_i) = \begin{cases} \dfrac{\max(0, \min(s_c, s_t)) - \max(0, \min(s_c, s_t)) + s_t}{2s_t} & s_c + s_i \leq 2s_t \\ \dfrac{s_c}{s_c + s_i} & s_c + s_i > 2s_t \end{cases} \tag{12}$$

$$p_i(s_c, s_i) = 1 - p_c\left(s_c - p_i\right) \tag{13}$$

where s_t is the minimum "target" number of spikes. The purpose of s_t is to avoid assigning a too high or low fitness to agents that fire few spikes through their outputs. The agent takes damage at every time step and is given by

$$d(s_c, s_i) = d_c p_c(s_c, s_i) + d_i p_i(s_c, s_i) \tag{14}$$

Damaging is performed until the agent runs out of health points and 'dies'. Subsequently, the fitness score of the agent is calculated from the fitness function expressed in Eq. 11. The damage to the health points in a correct action d_c is 1, in an incorrect one d_i is 2. Therefore, correct actions result in a longer lifetime. The value for the minimum 'target' number of spikes s_t is 3 spikes.

In the control task of cart-pole balancing, the behavior of the mutable environment is different. A new environment is presented to the agent either after its failure or after the maximum number of environment iterations is reached. Moreover, the agents do not have health points. The fitness score is the normalization of the number of iterations that the agent survived after all environment conditions were executed.

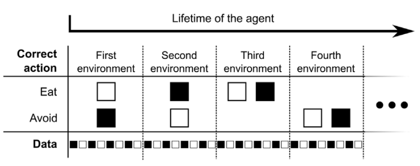


FIGURE 3
Example of the food foraging environment and how it progresses through the lifetime of the agent in a generation. The eight food samples per environment are illustrative. Our experiment uses four.

TABLE 1 Correct actions for all combinations of input food color and edible food in the food foraging task.

Food foraging environment conditions				
Edible	**Black**	**White**	**None**	**Both**
Input				
Black	Eat	Avoid	Avoid	Eat
White	Avoid	Eat	Avoid	Eat

4.3.1 Food foraging

The agent in the food foraging environment possesses just one light sensor for identifying the food "in front of it." There are two types of food: edible and poisonous. As such, food is represented in two colors: black and white. The environment changes by randomly defining which food color is edible or poisonous. In this environment, the agent can act in two ways: eating or avoiding the food. The sample has a predefined time of exposure to the agent. An action is performed after the first spike and it continues for every time step in the environment simulation. After this exposure time, the food is replaced by a new one. The agent can only discover whether it is exposed to an edible or poisonous food by interacting with it. An incorrect action is defined as eating poisonous food, or avoiding edible food, while a correct action is defined as eating edible food or avoiding poisonous one. If the agent makes an incorrect action, it receives a penalty signal, from which the agent should learn over the generations that it represents pain, revulsion, or hunger. If the agent makes a correct action, it receives a reward signal, from which it should learn that it represents the pleasure of eating delicious food or recognizing that the food is poisonous. In Figure 3, the food foraging environment is illustrated, how the environment changes and provides new food samples. In our experiment, the change of the environment occurs after presenting four food samples to the agent. The first food sample type is chosen randomly and alternates in every sample change. In Table 1, the four combinations of edible and poisonous food for the white and black ones are shown. To evolve the spiking neural network for the food foraging task, the parameters of the genetic algorithm are the following: the population size is set to 100 individuals, and the number of generations is set to 1,000. This task was chosen because of its simplicity. In particular, it allows a virtual wheeled robot to forage for food using proximity sensors, such as in the related work of Stanley et al. (2003).

4.3.2 Logic gates

In this environment, the mutable environmental state is a two-input logic gate. The environment provides the agent with two binary inputs, i.e., 0's and 1's. The agent's task is to predict the correct output for the current logic gate given the current input. Similar to the food foraging environment, it receives a reward signal if it is currently predicting the correct output, and a penalty signal if it is currently predicting the wrong output.

In order to measure the generalizing properties of agents, we use two different sets of environments: a training environment, which is used in calculating the fitness score while running the evolutionary algorithm, and a test environment which has a fully disjoint set of possible environmental states. A full overview of the logic gates found in both the training and the test environments, as well as the truth values for all input and output combinations, are found in Table 2 and Table 3. The evolution of the spiking neural network is performed by a population of 100 individuals through 1,000 generations.

4.3.3 Cart-pole balancing

The cart-pole balancing is a well-known control task used as a benchmark problem in reinforcement learning. In this environment, there is a cart that moves when a force is applied to the left or to the right every time step. In the middle of the cart, there is a vertical pole connected to a non-actuated joint. The goal of this environment is to maintain the pole balanced upright by controlling the forces that move the cart. Moreover, the cart cannot move beyond the limits of the track. The observations available to the controller are the cart position, the cart velocity, the pole angle, and the pole angular velocity.

For training, we use poles of different sizes, which are 0.5 (default), 0.3, and 0.7. For testing, the sizes are 0.4, and 0.6. Those pole sizes are depicted in the Supplementary Material. Each size can run up to 200 environment iterations and it is repeated three times during training for promoting stable controllers. If there are no more environment iterations or the pole falls, the cart-pole environment restarts with the next pole size while using the same SNN or finishes when all pole sizes were executed. The fitness score is calculated using the number of iterations the pole kept balanced. Subsequently, it is normalized to values between 0 and 1. The evolution for this task occurs with a population size of 256 during 500 generations.

TABLE 2 Truth table showing the correct output for each training logic gate.

Training logic gates

Input		A	B	NOT A	NOT B	Only 0	Only 1	XOR	XNOR
A	B								
0	0	0	0	1	1	0	1	0	1
0	1	0	1	1	0	0	1	1	0
1	0	1	0	0	1	0	1	1	0
1	1	1	1	0	0	0	1	0	1

TABLE 3 Truth table showing the correct output for each testing logic gate.

Test logic gates

Input		AND	NAND	OR	NOR
A	B				
0	0	0	1	0	1
0	1	0	1	1	0
1	0	0	1	1	0
1	1	1	0	1	0

4.4 Data representation

The data type in a spiking neural network is a binary time series or a spike train. Because the agent senses and acts in the environment, such data must be converted from the sensors and to the actuators. The flow of spikes over time can be quantified as firing rate, which corresponds to a frequency, or the number of spikes per second. The firing rate is the data representation that is converted as inputs and outputs for the SNN. However, the input firing rate must be within a minimum and a maximum value. In our experiments, we use the value range [$5Hz$, $50Hz$]. The minimum and maximum value of the firing rate are simplified to a real number range [0, 1]. It is preferable that the data from the sensors has also a minimum and a maximum value. Otherwise, it will be necessary to clip sensory values or map the values to a desirable range.

In the binary classification tasks, all inputs and outputs are binary. Therefore, the minimum and maximum values for the input firing rate stand for, respectively, 0 and 1, or *False* and *True*. To avoid having a predefined threshold firing rate for the output neurons, we opt to have two output neurons for one binary value. The neuron with the highest firing rate within the actuator time window is the one defining the binary output value. If these two output neurons have the same firing rate, then the last one with

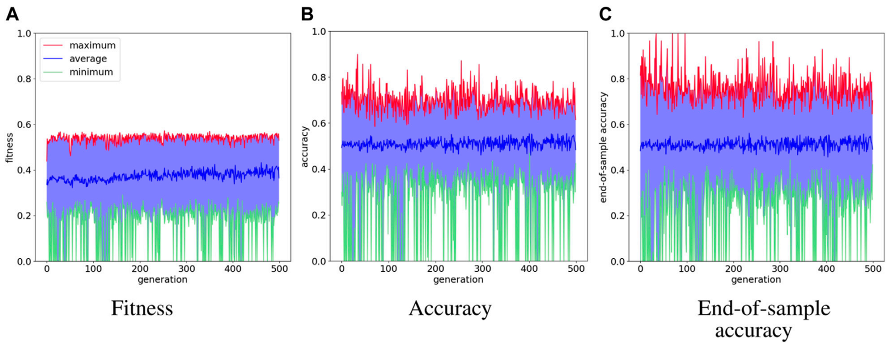


FIGURE 4
Evolution history of food foraging environment showing the average, minimum and maximum per generation.

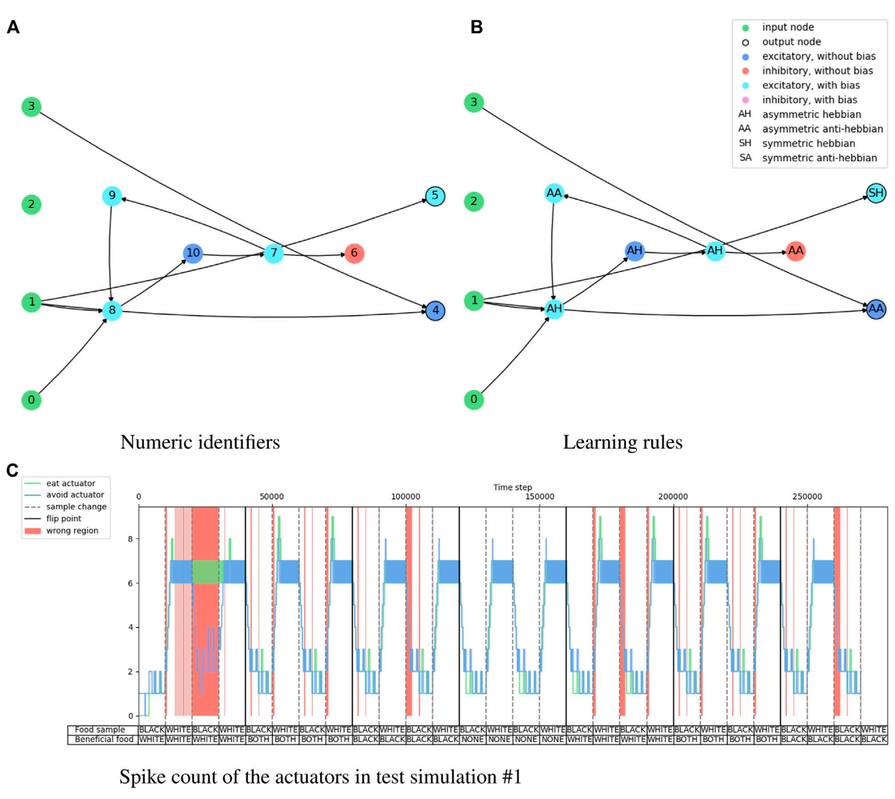

A
B
Numeric identifiers
Learning rules
input node
output node
excitatory, without bias
inhibitory, without bias
excitatory, with bias
inhibitory, with bias
AH asymmetric hebbian
AA asymmetric anti-hebbian
SH symmetric hebbian
SA symmetric anti-hebbian
C
eat actuator
avoid actuator
sample change
flip point
wrong region
Time step
Food sample
Beneficial food
BLACK WHITE BLACK WHITE BLACK WHITE BLACK WHITE BLACK WHITE BLACK WHITE BLACK WHITE BLACK WHITE BLACK WHITE BLACK WHITE BLACK WHITE BLACK WHITE BLACK WHITE BLACK WHITE
WHITE WHITE WHITE WHITE BOTH BOTH BOTH BOTH BLACK BLACK BLACK BLACK NONE NONE NONE NONE WHITE WHITE WHITE WHITE BOTH BOTH BOTH BOTH BLACK BLACK BLACK BLACK
Spike count of the actuators in test simulation #1

FIGURE 5
Illustration of the network topology and behavior of the highest accuracy agent in the food foraging task. The one-hot encoded input sample goes into nodes 0 and 1, the reward signal in node 2, and the penalty signal goes into node 3. Node 4 is the output for the "eat" actuator and node 5 is the output for the "avoid" actuator.

the highest value is selected. We also decided to have the same "two neurons-one binary value" strategy with the inputs, which consists of 0 or *False* being 01 in one-hot encoding, then (*low*, *high*) in firing rate, while 1 or *True* is 10 in one-hot encoding, so the firing rate is (*high*, *low*).

For the cart-pole control task, the inputs are real numbers, and the left and right actions are represented as two output neurons, similar to the outputs of the binary classification tasks. In this environment, the inputs are the cart position, cart velocity, pole angle, and pole angular velocity. Because we infer that real numbers converted to the firing rate of one neuron can be difficult to deal with in an adaptive spiking neural network (as also mentioned in Ref. (Pontes-Filho and Liwicki, 2019)), we decided to have three neurons for each input. The firing rate of the three neurons is similar to the sensitivity for the light spectrum of the three cone cells in the human eye (Bowmaker and Dartnall, 1980). We use the sigmoid function (Han and Moraga, 1995) for neurons #1 and #3 and a normalized version of the Gaussian function (Patel and Read, 1996) for neuron #2. The sigmoid equation is

$$\mathcal{F}_{sigmoid}(x \mid \omega, z, h, l) = \frac{h}{1 + e^{-\omega(x-z)}} + l, \qquad (15)$$

where x is the observation value from the environment, ω is the weight that adjusts the smoothness of the interval between 0 and

TABLE 4 Test simulations of the highest accuracy agent in the food foraging experiment. “Acc.” stands for accuracy and “EOS Acc.” for end-of-sample accuracy.

Food foraging test simulations

#	Acc. (%)	EOS Acc. (%)	Input order	Environment order
1	88.0	92.6	black, white	white, both, black, none
2	90.6	100	white, black	white, none, both, black
3	91.3	100	black, white	white, both, none, black
4	85.4	92.3	white, black	white, black, both, none
5	89.5	96.3	white, black	both, none, white, black
6	89.2	100.0	black, white	both, white, black, none
7	87.7	92.6	black, white	white, black, none, both
8	84.9	92.6	black, white	black, both, white, none
9	89.8	100	black, white	white, black, both, none
10	88.4	92.6	white, black	black, none, white, both
Avg	88.4	95.9	n/a	

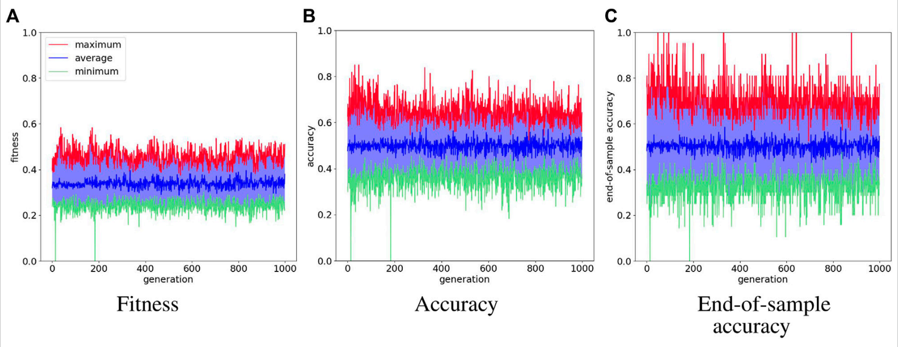


FIGURE 6
Evolution history of logic gate environment showing the average, minimum and maximum per generation.

1, z is the shift coefficient to adjust the function on the horizontal axis, h is the highest firing rate possible applied to an input neuron, and l is the lowest firing rate possible. The Gaussian function for converting observation value to firing rate is expressed by

$$\mathcal{F}_{Gaussian}(x \mid \mu, \sigma, h, l) = he^{\frac{-\left((x-\mu)^2\right)}{2\sigma^2}} + l, \qquad (16)$$

where μ is the mean and σ is the standard deviation. We replace $\frac{1}{\sigma\sqrt{2\pi}}$ in the original Gaussian function to h because, in this way, we can define the highest firing rate when the observation value is the mean. Neurons #1 and #3 use $\mathcal{F}_{sigmoid}$, while neuron #2 uses $\mathcal{F}_{Gaussian}$. The parameters and the figures with the illustration of those equations are included in the Supplementary Material.

5 Results

The evolution of the spiking neural networks in NAGI is evaluated with fitness score, accuracy, and end-of-sample accuracy for the binary classification tasks, which are food foraging and logic gate. The accuracy is measured at every time step of the simulation. The end-of-sample accuracy stands for the accuracy measured in the last time step of a

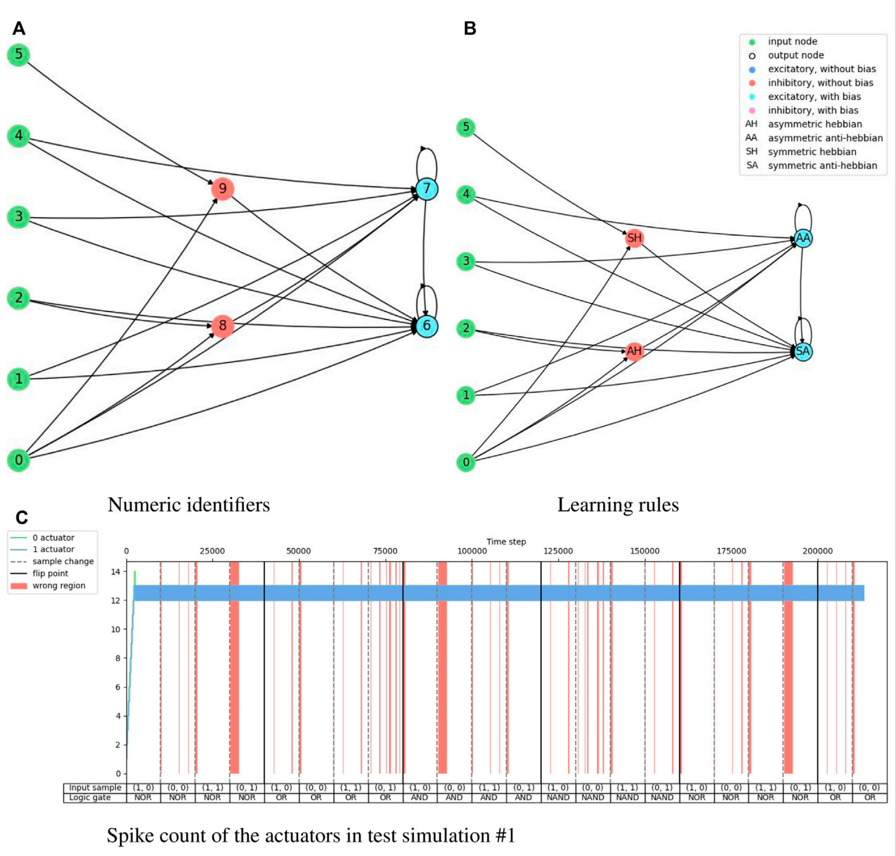


FIGURE 7
Illustration of the network topology and behavior of the highest training accuracy agent in the logic gate task. The one-hot encoded input sample "**(A)**" goes into nodes 0 and 1, the one-hot encoded input sample "**(B)**" goes into nodes 2 and 3, the reward signal goes into node 4, and the penalty signal into node 5. Node 6 is the output for the "0" actuator and node 7 is the output for the "1" actuator.

sample. The assessment performed for the control task with cart-pole balancing is done with the fitness score. We test the best performing agent in a task with ten simulations where their details are also provided.

Figure 4 shows the evolution history of the food foraging task. The average fitness score has a slight increase, but the maximum fitness score does not follow this trend. The accuracy and end-of-sample accuracy have high variation with their maximum values, but they consist of high accuracies. Moreover, some early generations register 100% end-of-sample accuracy. The three measurements do not improve through the generations. However, good solutions are already found in the first generation. Therefore, this is an easy task that requires a small SNN. For test simulations, we select the individual with the highest accuracy, which is found in generation number 34 and has an accuracy of 89.8%. Its fitness score is 0.541395 and its end-of-sample accuracy is 100%. Its topology is shown in Figure 5. Paying attention to this topology, the hidden nodes are not

TABLE 5 Test simulations of the highest training accuracy agent in the logic gate experiment. "Acc." stands for accuracy and "EOS Acc." for end-of-sample accuracy.

Logic gate test simulations

#	Acc. (%)	EOS Acc. (%)	Input order (A, B)	Environment order
1	89.8	100	(1, 0), (0, 0), (1, 1), (0, 1)	NOR, OR, AND, NAND
2	85.2	95.2	(1, 1), (1, 0), (0, 0), (0, 1)	OR, NOR, NAND, AND
3	86.0	100	(1, 0), (1, 1), (0, 1), (0, 0)	NOR, OR, AND, NAND
4	85.9	95.2	(0, 0), (1, 1), (0, 1), (1, 0)	NAND, AND, OR, NOR
5	79.9	85.7	(0, 0), (0, 1), (1, 0), (1, 1)	NAND, AND, NOR, OR
6	88.8	100	(1, 0), (0, 0), (1, 1), (0, 1)	AND, NAND, OR, NOR
7	85.1	90.5	(0, 0), (1, 1), (1, 0), (0, 1)	OR, NOR, NAND, AND
8	84.8	90.5	(1, 1), (0, 1), (0, 0), (1, 0)	NOR, NAND, OR, AND
9	83.7	85.7	(0, 0), (1, 0), (0, 1), (1, 1)	NAND, NOR, OR, AND
10	88.5	100	(1, 1), (1, 0), (0, 0), (0, 1)	NOR, AND, OR, NAND
Avg	85.7	94.2	n/a	

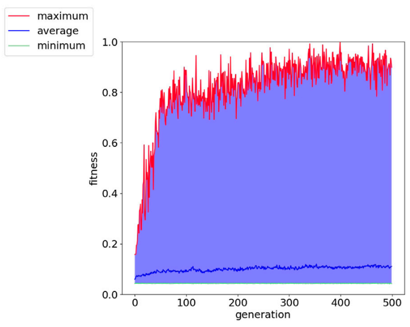


FIGURE 8
Fitness history of cart-pole balancing environment showing the average, minimum and maximum per generation.

TABLE 6 Test simulations of the highest fitness agent in the cart-pole balancing experiment.

Cart-pole balancing test simulations

#	Fitness	# Steps 0.4	# Steps 0.6	Environment order
1	1.000	200	200	0.4, 0.6
2	1.000	200	200	0.4, 0.6
3	1.000	200	200	0.6, 0.4
4	0.943	200	177	0.4, 0.6
5	0.800	154	166	0.6, 0.4
6	0.792	179	138	0.4, 0.6
7	0.835	200	134	0.6, 0.4
8	0.845	200	138	0.4, 0.6
9	0.873	200	149	0.6, 0.4
10	0.720	88	200	0.6, 0.4
Avg	0.874	178.3	171.5	n/a

needed. They form a loop that does not connect with the output nodes. The topology summarizes in one of the one-hot encoded input nodes (node 1) connecting to the two output nodes. Then, the node with the penalty signal (node 3) connects only with the node for the "eat" actuator (node 4). The behavior of the network is illustrated in Figure 5C. The topology of the network indicates that the two output neurons have the same data input from node 1, but the neuron for "avoid" action has a bias, which gives it a small excitatory current. If "avoid" is the wrong action, the penalty input signal from node 3 excites the output neuron for the "eat" action. This is how the spiking neural network decides the actions from "understanding" the feedback of the environment given by the penalty input signal. The result of the ten test simulations is presented in Table 4.

Figure 6 shows the training results of the logic gate task and it includes the test of the maximum individual of the measurement in every generation. The fitness score, accuracy, and end-of-sample accuracy maintain average values with high variation. However, the evolution of the agents in the logic gate task is similar to the one in the food foraging. The early generations already contain good spiking neural networks for the task. The best-performing agent is selected from the accuracy measurement. This individual is in generation 48 and has an accuracy of 85.0%. Its fitness score is 0.4421625 and its end-of-sample accuracy is 100%. The topology of this spiking neural

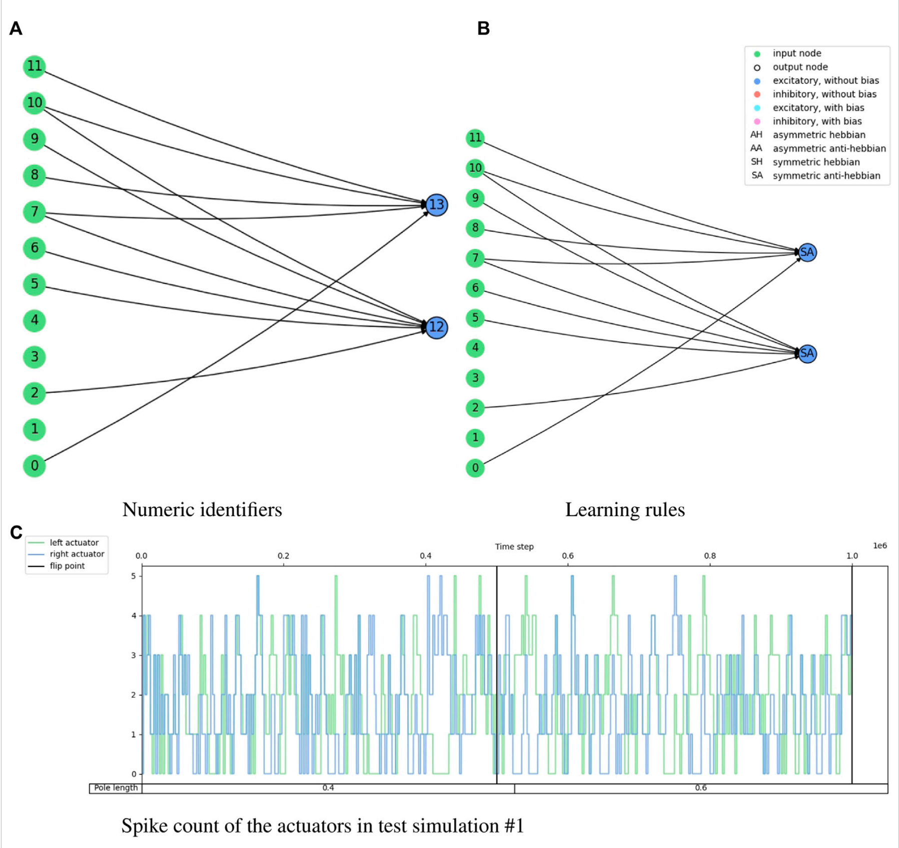


FIGURE 9
Illustration of the network topology and behavior of the highest training fitness agent in the cart-pole balancing experiment. The 3-tuple of input nodes (0, 1, 2) gets the converted firing rate from the observation of the cart position, (3, 4, 5) from the cart velocity, (6, 7, 8) from the pole angle, and (9, 10, 11) from the pole angular velocity. Node 12 is the output for the 'left' action and node 13 is the output for the "right" action.

network is shown in Figure 7. Its behavior is shown in Figure 7C. Even though we have trained with a "confidence" factor in the fitness function, the spike counts are still with almost the same values. Table 5 contains the accuracy and end-of-sample accuracy of ten test simulations, which indicates that the SNN can be general to reproduce the behavior of logic gates without being trained to them.

Figure 8 shows the fitness score history through the evolution for the cart-pole balancing task. This task is the one with the highest difficulty to find a good genome for the adaptive spiking neural network. It can be noted that the fitness score improves through the generations. The maximum fitness score in a generation goes from around 0.16 in the first generation to 0.99944 in generation number 399. Such an individual is the one selected for the test simulations. Its topology is illustrated in Figure 9 and the spike counts of the actuators for "left" and "right" actions are shown in Figure 9C. The spiking neural network has no hidden neurons. Therefore, the SNN works as an input selection for the output neurons. The result of the ten test simulations is presented in Table 6. When the pole is balanced for more than 100 iterations, the controller is considered successful.

6 Discussion and conclusion

We successfully solved all three presented tasks with the NAGI framework. The spiking neural networks found showed generality to the binary classification tasks, even to unseen conditions in the case of the emulation of logic gates. The neuroevolution produced rather simple topologies for the SNNs. We infer that binary classification is easy due to the binary performance feedback. For further research, multi-class classification is considered.

The cart-pole balancing task was successfully solved without any hidden neurons. The conversion of one observation into three input neurons is used to avoid the requirement of weight fine-tuning due to small differences in firing rate and also to the assumption that Hebbian plasticity works better with binary data (active and inactive) (Pontes-Filho and Liwicki, 2019). With such a conversion, the SNN became an input selection.

The topologies for the three tasks caught our attention because almost all output excitatory neurons were anti-Hebbian, and the two inhibitory hidden neurons in the logic gate solution have Hebbian neuroplasticity. Our initial hypotheses were that excitatory neurons mainly have Hebbian learning rules, and inhibitory neurons are anti-Hebbian. That was the reason for having different probabilities for anti-Hebbian and Hebbian learning rules depending on the type of the neurotransmitter when adding a new neuron through mutation.

Even though there is elitism, the performance measurements are unstable through generations. This is a demonstration of the randomness in the initialization of the weights, and input and environment order. This can be perceived in the results of the ten test simulations of the three tasks.

For future work, we plan to attempt more challenging tasks. If there is a failure in executing the task, the constraints imposed on NAGI can be eased. A major constraint is that one neuron has one plasticity rule for all dendrites. Maybe its removal can simplify issues in difficult tasks. This constraint was intended to reduce the dimensionality of the search space in the neuroevolution and an assumption that the dendrites in the same neuron adapt under one learning rule. This modification is also aligned with the work of Najarro and Risi, (2020), which has meta-learning properties for more difficult control tasks than the cart-pole balancing, such as top-down car racing and quadruped walk. Another opportunity is the addition of curriculum learning (Bengio et al., 2009; Narvekar et al., 2020) for increasing the complexity of the task while the agent becomes better over the generations.

Data availability statement

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

Author contributions

SP-F had the main idea, supervised most part of the work, implemented additional experiments, and wrote the initial draft of the manuscript. KO contributed to the writing of the initial draft and performed most of the experimental work as part of his master's thesis while being mainly supervised by SP-F. AY, MR, PH, and SN co-supervised the work. All authors reviewed the manuscript.

Funding

This work was partially funded by the Norwegian Research Council (NFR) through their IKTPLUSS research and innovation action under the project Socrates (grant agreement 270961).

Acknowledgments

This manuscript includes materials from the Master's thesis of Olsen, (2020). Its preprint is available in Ref. (Pontes-Filho et al., 2022).

Conflict of interest

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

Publisher's note

Supplementary material

The Supplementary Material for this article can be found online at: https://www.frontiersin.org/articles/10.3389/frobt.2022.1007547/full#supplementary-material

References

Ardiel, E. L., and Rankin, C. H. (2010). An elegant mind: Learning and memory in caenorhabditis elegans. *Learn. Mem.* 17, 191–201. doi:10.1101/lm.960510

Bengio, Y., Louradour, J., Collobert, R., and Weston, J. (2009). "Curriculum learning," in Proceedings of the 26th annual international conference on machine learning, 41–48.

Betts, J. G., Young, K. A., Wise, J. A., Johnson, E., Poe, B., Kruse, D. H., et al. (2013). *Anatomy and physiology*. Houston, TX: OpenStax. Available at: https://openstax.org/details/books/anatomy-and-physiology.

Bowmaker, J. K., and Dartnall, H. J. (1980). Visual pigments of rods and cones in a human retina. *J. Physiology* 298, 501–511. doi:10.1113/jphysiol.1980.sp013097

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., et al. (2016). Openai gym. *arXiv preprint arXiv:1606.01540*

Camuñas-Mesa, L. A., Linares-Barranco, B., and Serrano-Gotarredona, T. (2019). Neuromorphic spiking neural networks and their memristor-cmos hardware implementations. *Materials* 12, 2745. doi:10.3390/ma12172745

Crosby, M., Beyret, B., and Halina, M. (2019). The animal-ai olympics. *Nat. Mach. Intell.* 1, 257. doi:10.1038/s42256-019-0050-3

Diehl, P., and Cook, M. (2015). Unsupervised learning of digit recognition using spike-timing-dependent plasticity. *Front. Comput. Neurosci.* 9, 99. doi:10.3389/fncom.2015.00099

Doncieux, S., Bredeche, N., Mouret, J.-B., and Eiben, A. E. G. (2015). Evolutionary robotics: What, why, and where to. *Front. Robot. AI* 2, 4. doi:10.3389/frobt.2015.00004

Funes, P., and Pollack, J. (1998). Evolutionary body building: Adaptive physical designs for robots. *Artif. Life* 4, 337–357. doi:10.1162/106454698568639

Gaier, A., and Ha, D. (2019). "Weight agnostic neural networks," in *Advances in Neural Information Processing Systems*. Editors H. Wallach, H. Larochelle, A. Beygelzimer, F. D'. Alché-Buc, E. Fox, and R. Garnett (Curran Associates, Inc.) 32. Available at: https://proceedings.neurips.cc/paper/2019/file/e98741479a7b998f88b8f8c9f0b6b6f1-Paper.pdf.

Han, J., and Moraga, C. (1995). "The influence of the sigmoid function parameters on the speed of backpropagation learning," in Proceedings of the International Workshop on Artificial Neural Networks: From Natural to Artificial Neural Computation (London, UK, UK: Springer-Verlag), 195–201.

Hebb, D. O. (1949). *The organization of behavior: A neuropsychological theory*. New York: Wiley.

Holland, J. H. (1992). Genetic algorithms. *Sci. Am.* 267, 66–72. doi:10.1038/scientificamerican0792-66

Izhikevich, E. M. (2003). Simple model of spiking neurons. *IEEE Trans. Neural Netw.* 14, 1569–1572. doi:10.1109/tnn.2003.820440

Kulik, Y., Jones, R., Moughamian, A. J., Whippen, J., and Davis, G. W. (2019). Dual separable feedback systems govern firing rate homeostasis. *Elife* 8, e45717. doi:10.7554/elife.45717

Lake, B. M., Ullman, T. D., Tenenbaum, J. B., and Gershman, S. J. (2017). Building machines that learn and think like people. *Behav. Brain Sci.* 40, e253. doi:10.1017/s0140525x16001837

Langton, C. (2019). *Artificial life: Proceedings of an interdisciplinary workshop on the synthesis and simulation of living systems*. London: Routledge.

Li, Y., Zhong, Y., Zhang, J., Xu, L., Wang, Q., Sun, H., et al. (2014). Activity-dependent synaptic plasticity of a chalcogenide electronic synapse for neuromorphic systems. *Sci. Rep.* 4, 4906. doi:10.1038/srep04906

Liu, Y.-H., and Wang, X.-J. (2001). Spike-frequency adaptation of a generalized leaky integrate-and-fire model neuron. *J. Comput. Neurosci.* 10, 25–45. doi:10.1023/a:1008916026143

Mautner, C., and Belew, R. K. (2000). Evolving robot morphology and control. *Artif. Life Robot.* 4, 130–136. doi:10.1007/bf02481333

Nadji-Tehrani, M., and Eslami, A. (2020). A brain-inspired framework for evolutionary artificial general intelligence. *IEEE Trans. Neural Netw. Learn. Syst.* 31, 5257–5271. doi:10.1109/tnnls.2020.2965567

Najarro, E., and Risi, S. (2020). "Meta-Learning through Hebbian Plasticity in Random Networks," in *Advances in Neural Information Processing Systems*. Editors H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin (Curran Associates, Inc.) 33, 20719–20731. Available at: https://proceedings.neurips.cc/paper/2020/file/ee23e7ad9b473ad072d57aaa9b2a5222-Paper.pdf.

Narvekar, S., Peng, B., Leonetti, M., Sinapov, J., Taylor, M. E., and Stone, P. (2020). Curriculum learning for reinforcement learning domains: A framework and survey. *J. Mach. Learn. Res.* 21 (181), 1–50.

Olsen, K. (2020). *Neuroevolution of artificial general intelligence*. Oslo: University of Oslo.

Patel, J. K., and Read, C. B. (1996). *Handbook of the normal distribution*, 150. CRC Press.

Pontes-Filho, S., and Liwicki, M. (2019). "Bidirectional learning for robust neural networks," in 2019 International Joint Conference on Neural Networks (IEEE), IJCNN '19, 1–8.

Pontes-Filho, S., and Nichele, S. (2019). A conceptual bio-inspired framework for the evolution of artificial general intelligence. *arXiv preprint arXiv:1903.10410*

Pontes-Filho, S., Olsen, K., Yazidi, A., Riegler, M., Halvorsen, P., and Nichele, S. (2022). Towards the neuroevolution of low-level artificial general intelligence. *arXiv preprint arXiv:2207.13583*

Randi, F., and Leifer, A. M. (2020). Measuring and modeling whole-brain neural dynamics in caenorhabditis elegans. *Curr. Opin. Neurobiol.* 65, 167–175. doi:10.1016/j.conb.2020.11.001

Reed, S., Zolna, K., Parisotto, E., Colmenarejo, S. G., Novikov, A., Barth-Maron, G., et al. (2022). A generalist agent. *arXiv preprint arXiv:2205.06175*

Risi, S., and Stanley, K. O. (2010). "Indirectly encoding neural plasticity as a pattern of local rules," in International Conference on Simulation of Adaptive Behavior. Springer, 533–543.

Risi, S. (2021). The future of artificial intelligence is self-organizing and self-assembling. Available at: sebastianrisi.com

Shen, Z., Liu, J., He, Y., Zhang, X., Xu, R., Yu, H., et al. (2021). Towards out-of-distribution generalization: A survey. *arXiv preprint arXiv:2108.13624*

Stanley, K. O., Bryant, B. D., and Miikkulainen, R. (2003). Evolving adaptive neural networks with and without adaptive synapses. In The 2003 Congress on Evolutionary Computation (IEEE), 2557–2564.

Stanley, K. O., Clune, J., Lehman, J., and Miikkulainen, R. (2019). Designing neural networks through neuroevolution. *Nat. Mach. Intell.* 1, 24–35. doi:10.1038/s42256-018-0006-z

Stanley, K. O., and Miikkulainen, R. (2002). Evolving neural networks through augmenting topologies. *Evol. Comput.* 10, 99–127. doi:10.1162/106365602320169811

Sukenik, N., Vinogradov, O., Weinreb, E., Segal, M., Levina, A., and Moses, E. (2021). Neuronal circuits overcome imbalance in excitation and inhibition by adjusting connection numbers. *Proc. Natl. Acad. Sci. U. S. A.* 118, e2018459118. doi:10.1073/pnas.2018459118

Tavanaei, A., Ghodrati, M., Kheradpisheh, S. R., Masquelier, T., and Maida, A. (2019). Deep learning in spiking neural networks. *Neural Netw.* 111, 47–63. doi:10.1016/j.neunet.2018.12.002

Taylor, T. (2019). Evolutionary innovations and where to find them: Routes to open-ended evolution in natural and artificial systems. *Artif. Life* 25, 207–224. doi:10.1162/artl_a_00290

Thrun, S., and Pratt, L. (1998). "Learning to learn: Introduction and overview," in *Learning to learn* (Springer), 3–17.

Trappenberg, T. (2009). *Fundamentals of computational neuroscience*. Oxford: OUP Oxford.

Watson, R. A., Ficiei, S. G., and Pollack, J. B. (1999). "Embodied evolution: Embodying an evolutionary algorithm in a population of robots," in Proceedings of the 1999 Congress on Evolutionary Computation-CEC99 (Cat. No. 99TH8406).

Welleck, S. (2019). Evolving networks. Available at: wellecks.wordpress.com

Zador, A. M. (2019). A critique of pure learning and what artificial neural networks can learn from animal brains. *Nat. Commun.* 10, 3770–3777. doi:10.1038/s41467-019-11786-6

Zohora, F. T., Karia, V., Daram, A. R., Zyarah, A. M., and Kudithipudi, D. (2021). "Metaplasticnet: Architecture with probabilistic metaplastic synapses for continual learning," in 2021 IEEE International Symposium on Circuits and Systems (ISCAS) (IEEE).

Paper C.1

Towards self-organized control: using neural cellular automata to robustly control a cart-pole agent
(Variengien et al., 2021)

Author(s):
Alexandre Variengien, Sidney Pontes-Filho, Tom Glover and Stefano Nichele

Published in journal:
Innovations in Machine Intelligence (IMI), vol. 1, pp. 1-14, 2021

Paper C.1
Variengien et al. (2021)

Innovations in Machine Intelligence (IMI), 2021, 1–14

doi: 10.54854/imi2021.01

RESEARCH PAPER

Towards Self-organized Control: Using Neural Cellular Automata to Robustly Control a Cart-pole Agent

Alexandre Variengien[1,2,3], Sidney Pontes-Filho[1,4], Tom Eivind Glover[1] and Stefano Nichele[1,2,*]

[1]Department of Computer Science, Oslo Metropolitan University, Oslo, Norway
[2]Department of Holistic Systems, Simula Metropolitan Centre for Digital Engineering, Oslo, Norway
[3]Department of Computer Science, Ecole Normale Superieure de Lyon, Lyon, France
[4]Department of Computer Science, Norwegian University of Science and Technology, Trondheim, Norway

[*]Corresponding author: stenic@oslomet.no

Abstract

Neural cellular automata (Neural CA) are a recent framework used to model biological phenomena emerging from multicellular organisms. In these systems, artificial neural networks are used as update rules for cellular automata. Neural CA are end-to-end differentiable systems where the parameters of the neural network can be learned to achieve a particular task. In this work, we used neural CA to control a cart-pole agent. The observations of the environment are transmitted in input cells while the values of output cells are used as a readout of the system. We trained the model using deep-Q learning where the states of the output cells were used as the Q-value estimates to be optimized. We found that the computing abilities of the cellular automata were maintained over several hundreds of thousands of iterations, producing an emergent stable behavior in the environment it controls for thousands of steps. Moreover, the system demonstrated life-like phenomena such as a developmental phase, regeneration after damage, stability despite a noisy environment, and robustness to unseen disruption such as input deletion.

Key words: Neural Cellular Automata; Developmental AI; Self-Organised Control; Differentiable Self-Organisation; Reinforcement Learning

Introduction

One of the most remarkable feats of life is the developmental process leading to the emergent complexity of the human brain from a single cell. The field of neurodevelopment (i.e., the development of the nervous system) has been investigating this problem for decades. These studies led to the discovery of the intricate mechanisms by which gradients of chemicals and local cell interactions shape the differentiation of pre-neural cells and the organization of their connections [1]. Even after the brain is considered fully grown, its developmental process does not stop. New neurons are formed continuously until death, and the shape and the strength of their connections change. Such neural plasticity is influenced by the sensory inputs received by the individual and is considered to be at the root of the emergence of intelligence. In addition to the ability to cope with a changing environment, plasticity provides remarkable robustness. For example, after a stroke, the neural network reorganizes in a new architecture to preserve motor function [2]. It can also adapt to sensory deprivation to extract the most information from the remaining senses. This is the case in blind individuals: the processing of auditive information can be partially deferred to the visual cortex, improving their sound localization abilities [3].

Neural plasticity can be considered as a part of the whole mechanism governing homeostasis of both shape and function. This conceptual proximity is also justified by biological evidence such as the discovery of the role of electrical activity during morphogenesis. In particular, the same ion channels are used both for local communication between non-neuronal cells during embryogenesis and in the neurons to carry action potentials [4]. More generally, there seems to exist a continuum between the phenomena we usually call growing, learning, and computing. Each of these abilities

Received: 16 Jul 2021, Accepted: 20 October 2021, Published online: 30 December 2021.

may be considered a different aspect of the same underlying self-organizing system.

Moreover, there is evidence that the DNA does not encode precise details of the resulting neural network. There is an information gap between the size of the DNA and the complexity of the neural network, generally referred to as a genomic bottleneck [5]. The DNA only specifies the local behavior of cells through the shape of the proteins it encodes. The neural network is then a structure that emerges through these local interactions and yields useful biological processing of the inputs received by the senses [6].

Despite the crucial role of growth in the emergence of intelligence, modern advances in artificial neural networks mainly focus on the handcrafted design of static maps of neural connections. During the phase called learning—that is in fact quite far from the biological sense of this word [5] —the connections of this architecture are optimized to reduce the error on the task to solve.

Some effort has been made to include an automatic process to incrementally design neural network architectures. These techniques include the use of genetic algorithms [7][8] or the introduction of a growing phase in artificial neural networks [9] [10]. Nonetheless, in these works, the process is a tool for navigating the topological parameter space, rather than treating the learning as a developmental problem.

The developmental problem has also been addressed as an independent task. In this case, the goal is to model the phenomena of morphogenesis observed in living organisms. Successful approaches used cellular automata along with artificial neural networks implementing local rules. They were able to produce complex patterns from localized interactions [11] [12].

More radical approaches tried to develop an artificial substrate where life-like phenomena could emerge in an open-ended environment [13] [14]. Despite their great promise, we are still far from recreating the whole evolution process that gives rise to intelligence.

In this work, we attempt at bridging the gap between growth and computation by using neural cellular automata (neural CA) [12]. Neural CA are spatially distributed systems composed of cells that interact through local interaction. Their update rule consists of an artificial neural network and thus can be optimized through classical and efficient gradient descent-based techniques. Even if the learning process is still happening through a procedure exterior to the system, the local rules encode both the developmental process—the transformation from a random grid to a configuration suitable for computation—and the information processing itself. We found that the cells were able to transmit and combine in a meaningful way the information from input cells to output cells used as a readout. The system demonstrates long-term stability and robustness to noise and damage. We illustrate these abilities on a simple control task as a proof of concept. Despite the fact that, in its current form, the system cannot compete with traditional techniques in terms of learning efficiency, this approach is justified by the future opportunities it opens. We discuss the potential future work in the last section of the paper.

Related works

The idea that a controller can emerge from a self-organizing system is not new. One of the most studied examples concerns the gait transition in animals i.e. going from walking to running when the velocity of the motion increases. The different limb coordination strategies observed during each gait do not seem to be the result of a control plan transmitted by the brain. Instead, this phenomenon has been described as a phase transition in the self-organizing system composed of the bones, nerves, and muscles used for locomotion [15]. This has notably been modeled by coupled differentiable equations describing mechanical dynamics and neural oscillators to reproduce walking motion [16].

Other techniques such as dynamic neural fields have also been used. Their dynamics have been theoretically described in [17]. This enables the design of systems that exploit their emerging patterns of activation for human-robot interaction [18].

More generally, it has been argued that there exists close proximity between goal-directed behavior relying on feedback loops where the agent tries to adjust its action to minimize the distance to its desired state and the dynamics of self-organizing systems. These two models could be different ways of thinking about the same processes [19].

Goal-directed cellular automata

The artificial design of self-organizing systems has been strongly focused on cellular automata (CA) because of their simplicity and their general abilities. CA have been historically introduced to address general questions about multicellularity in life: how can complex shapes be created from a single cell, maintained, and then replicated?

While the first works focused on handcrafted rules to create self-replicating systems [20], more recent projects complexified the rules updating the cell states. To search among the wide rule spaces, genetic algorithms have been frequently used to find CA that exhibited a predefined behavior. This enables the design of CA that robustly grows a shape, in effect exhibiting homeostasis [21]. Another work was able to develop a targeted shape and maintain it despite damage [22].

Instead of a classical look-up table, some works used update rules implementing more complex algorithms. These types of update rules were used as a generalization of evolvable circuits [23] to design CA that performs tasks broader than the historical goal of CA [24]. As in this case, CA have been used more generally not only for questions related to shape but also for useful decentralized computation [25].

To improve the search with genetic algorithms and favor evolvability, some works used variable genotype size [26]. Other works also included developmental function in the CA rules in order to approach the fuzzy, one-to-many, function that maps a genotype and an environment to a phenotype. This was done through the addition of self-modifying abilities in the code of each cell, leading to the creation of self-replicating systems [27].

Neural cellular automata

Precedent works used evolved neural networks to create CA that grow desired shapes [11]. Then, the introduction of neural CA [12] allowed the optimization of the neural networks used as update rules using the language of differentiable programming instead of genetic algorithms.

This model adds further elements to the questions for which the CA were created. Neural CA enable the creation of self-repairing systems that can grow complex shape from a single cell in 2D [12] or in 3D [28], and regeneration of functional bodies such as soft robots [29]. Beyond investigating homeostasis of shape, neural CA have also been used for decentralized pattern recognition [30] as well as texture synthesis [31].

Method

The pole balancing task

The cart-pole problem is a commonly used toy problem in the reinforcement learning community. In this environment, an agent controls a cart-pole system. It can observe the pole angle and its angular velocity, the cart position, and its linear velocity. Based on these observations, the agent must decide whether to apply a

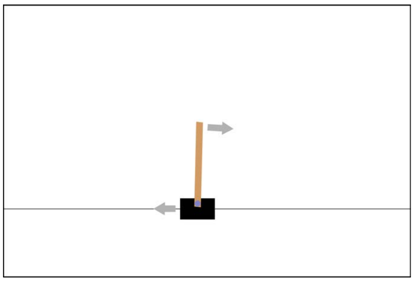


Figure 1. The cart-pole environment. The top arrow represents the angular velocity of the pole, the bottom arrow, the velocity of the cart.

force on the left or the right of the cart in order to maximize reward, i.e. the time spent with the pole up and the cart in the center. The simulation ends if the cart hits a wall or if the pole falls. We chose this problem because of its low number of inputs and outputs that allow the use of a small-sized grid and an easy experimentation environment.

We used the implementation of this environment provided by the OpenAI gym library[1]. We modified the reward function to favor agents that stay in the center. This modification made the behavior of avoiding walls emerge faster because the short-term maximization of the reward was aligned with its long-term maximization. We made the choice to change the reward function to speed up the training and to provide easier experimentation and replication. The reward given at time step t is given by the equation (1) where x is the position of the cart, L is the length of the track. In the center ($x = 0$), the agent receives 1, the maximum reward, while if it is on the edges of the track ($x \pm L$), the agent receives 0, the minimum reward.

$$r_t = \begin{cases} \cos\left(\frac{x\pi}{2L}\right)^2 & \text{if } t \text{ is not a final step} \\ -100 & \text{else.} \end{cases} \tag{1}$$

Cell state

There are 3 types of cells in a grid: the intermediate, the input, and output cells.

The state of each cell is composed of 6 channels. The first is the *information channel* where meaningful input and output information transit. The third is identifying the inputs: it is equal to 1 in the input cells, 0 elsewhere. The fourth similarly identifies the outputs. The remaining three are hidden channels.

The state of the input cells cannot be changed, the information channel transmits the observation from the environment, and the other channels except the output identifier are set to 1. The values of the information channel of the output cells are used as the output of the system to be optimized to solve the task.

The meaning of each channel and the different types of cells are represented in the figure 2.

Input encoding

We use redundancy in the inputs: each of the 4 physical observations of the environment is linked to 2 input cells. Because we have 4 types of observation, there are $\binom{4}{2} = 6$ possible pairs of observations. Once the input cells are arranged in a circle, there are $4 * R$ pairs of neighbors, where R is the amount of redundancy. We chose $R = 2$, this way, every pair of observations is present as a pair of neighbor input cells. This argument was introduced because we hypothesized that the closer two input cells are, the easier it will be to combine the information. Thus, we thought that this choice of position could improve the opportunity for information combination. Furthermore, redundancy could also provide more robustness: if an input cell is damaged, the information it holds is also contained in an undamaged input cell somewhere else.

Note that the type of information contained in the input is not directly provided. To know which observation each input cell encodes, the CA must rely on the spatial position of the inputs or the value of the information channel.

The value of each observation is scaled by a constant factor before being transmitted to the input cell. The choice of the factor corresponding to each observation was chosen to get similar ranges of values in the information channel.

Cell position

The 8 inputs are arranged on a circle to form an octagonal shape (dotted line) on a 32x32 grid with zeros at the boundaries as shown in figure 3. The two output cells are offset by 2 cells from the center of the octagon. We chose this configuration to ensure an almost equal distribution of the distance between each input and output. We hypothesized that the closer an input cell is to an output cell, the more the input will influence the output. The position of the input cell was then chosen to avoid any bias toward some inputs.

Model

Except for the design of the cell states, the neural CA architecture we used is similar to the one described for the self-classifying MNIST task [30]. The perception layer is composed of 20 learnable 3x3x6 filters, and the single hidden layer counts 30 units. In total, our model has 1854 learnable parameters.

As in the original model, the update rule is stochastic: at each step, each cell has a 0.5 probability of being updated. This choice is made to avoid temporal synchronization that relies on a centralized clock. The figure 4 summarizes the architecture of our model.

Training procedure

Our model can be abstracted as a black-box function that takes inputs (that will be fed to the information channel of input cells) and transforms them into outputs (the information channel of output cells). This function is differentiable with respect to its parameters (the neural network used as the update rule) and thus can be optimized with classical gradient descent techniques. In this case, we used the Adam optimizer [32] provided by the TensorFlow library[2]. This choice was made as this is also the optimizer successfully applied to neural CA in [12] and [30]. We did not perform any optimization of the training procedure (optimizer choice, hyperparameters, learning algorithm, etc) as this was beyond the scope of this paper. Our main aim was to provide a proof of concept.

Algorithm

To tackle the cart-pole problem, we used Deep Q-learning [33] where the usual artificial neural network is seamlessly replaced by a neural CA. The deep Q-learning algorithm aims at approaching the expected reward given a state and an action. More precisely, the

1 https://gym.openai.com/

2 https://www.tensorflow.org/

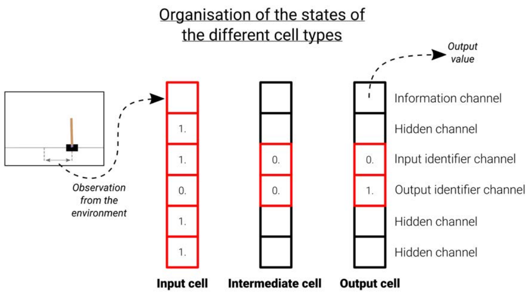


Figure 2. The role of the different channels and types of cells. Each cell is composed of a 6-dimensional vector, each dimension is called a channel. Different channels have different behaviors: some are fixed (in red) others can change according to the update function (in black). Moreover, channels depend on the cell type. Input cells have only constant channels except for their information channel that is defined by the observation of the environment. Excluding the identifier channels, intermediate cells are free to evolve and can influence neighboring cells. While output cells are similar to intermediate cells, they can be identified by the output identifier channel and the value of their information channel is used as a Q-value estimate.

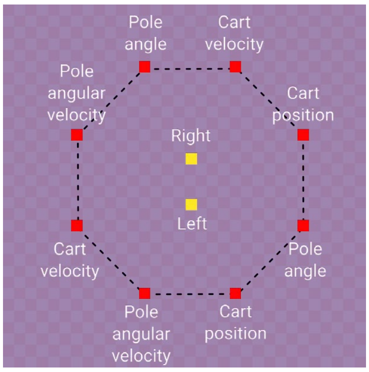


Figure 3. The position of input (in red) and output cells (in yellow).

function to be learned is given by equation (2) where t is the index of the current time-step. s_t is the current state and a_t the action to evaluate. r_t is the reward and γ is a discount factor.

$$Q(s_t, a_t) = E[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + ... | s_t, a_t] \tag{2}$$

To keep the CA values in the information channel in a range coherent with the other cells, we scale the outputs of the CA by a factor of 100 to get the Q-value estimates. The reason for the scaling is that the Q-value can be as low as -100 and as high as $1 + \gamma + \gamma^2 + ... = \frac{1}{1-\gamma}$ if the agent gets a reward of 1 for an infinite number of steps. In practice, we used $\gamma = 0.95$ so the higher bound of the Q-value is 20. The factor 100 was chosen to bound the prediction (in the case where they are close to the Q-value) to $[-1, 0.2]$: this is coherent with the overflow loss that penalizes values outside $[-5, 5]$, while keeping some margin.

Loss function

The loss function for the task is the L2 loss between the output and the target. To achieve long-term stability, we added a penalty for cells that have channel values out of bound [-5,5]. Note that excepted this overflow condition, the states of the intermediate cells are not directly optimized, they are free to evolve insofar as their influence on the outputs reduces the error. The formula used to compute the loss is given in equation (3) where N is the size of the grid, λ is a parameter to control the amount of overflow penalty, and clip is a function for limiting the values to the threshold interval [-5,5].

$$\text{Loss} = \text{L2}(outputs, target) + \frac{\lambda}{N^2} \sum_{i,j=1}^{N} \sum_{chan=1}^{6} (\text{clip}(Grid_{i,j,chan}, -5, 5) - Grid_{i,j,chan})^2 \tag{3}$$

Robustness

Damage

To increase the robustness of the system, we damage half of the grids present in the pool. Damage consists of a circle of the grid replaced by uniform random values in [-1,1], as shown in black in figure 5. Note that damage impacts all the channels that can be modified and that inputs are not affected by damage while outputs are.

Noise

Before applying each update, we perturbed it with uniform noise. Taking inspiration from what was done in [30], we used a noisy update to favor a long-term stabilization of the cell states. In total, the system has three layers of noise: the stochastic update where half of the cells are randomly chosen not to be updated, the damage of the grid, and the random perturbation of the values of the update.

Neural CA training

As introduced in [12] we used pool sampling for the states of the neural CA during training to learn persistent behavior.

As described in the deep-Q learning algorithm, we alternate phases where we explore the environment by letting the neural CA controls the cart-pole agent and by taking random actions; and training the neural CA using the target values based on the rewards stored in the memory of the agent.

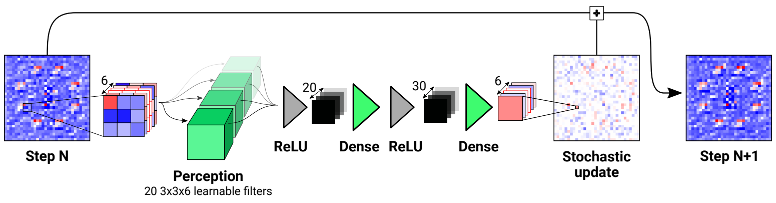


Figure 4. The architecture of our model. Positive values are depicted in red, negative in blue. Green objects identify the learnable parameters of the model.

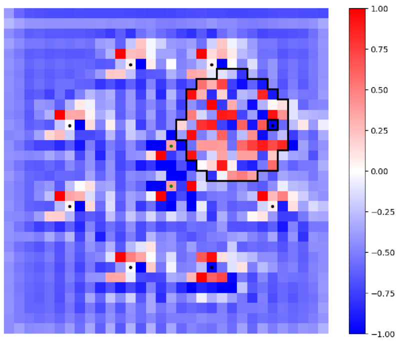


Figure 5. A damaged grid is shown on the information channel. The shape of the damaged region is an irregular circle due to rasterization effects.

Environment exploration

To get a stable long-term behavior of the cart-pole agent we did not use a fixed horizon for the environment. Instead, we use pool sampling also for the states of the cart-pole, as done for the neural CA grids.

The probability of taking a random action is given by the parameter ϵ that is decreased during the training of the agent, as in the original deep-Q learning algorithm. The exploration of the environment begins by sampling a grid from the grid pool, a cart-pole state from the pool of environment states. Then we let the neural CA, starting from the sampled grid, evolve for a random number of steps from 50 to 60. After we choose the action that corresponds to the greatest output of the neural CA, we obtain a new environment state. We put the grid back in the grid pool and sample a new one. We repeat this operation for K environment steps.

If the environment ends, we reset the environment to reach the end of the K steps. After the K steps, the state of the cart-pole is committed in the pool of environment states. We also randomly replace grids by the initial state to be sure that the neural CA always keeps the knowledge of how to start from a raw grid. The procedure is illustrated in the figure 6 and its pseudocode can be found in the algorithm 1.

Training

Between each exploration phase, we sample several batches of transitions (o_t, a_t, r_t, o_{t+1})—where o_t is the observation at time t, a_t the action taken, and r_t the reward received—from the memory of the agent that was stored during the exploration phases. We train the neural CA according to the expression of the target value and the error to optimize given by the deep-Q learning algorithm (equation (5)). Then, each transition is matched with a neural CA grid randomly sampled among the pool of grids that we let evolve for 50 to 60 steps. We next compute the loss on the final state of the grid, according to the formula (5) where y_i is the target value for a given transition at time j sampled from the memory of the agent, and $Q(o_j, a_j; \theta)$ is the output of the neural CA for an action a_j and an observation o_j. θ are the parameters of the update rule to be optimized. We perform a gradient step on a batch composed of 16 such grids. As described in [12], back-propagation through time is used to compute the gradient of the loss with respect to the parameters of the update rule. The pseudocode of the training phase is described in the algorithm 2. The exploration and training procedure are used in the algorithm 3 that describes the full optimization process of the neural CA.

- $$y_j = \begin{cases} r_j & \text{if } j \text{ is a final step} \\ r_j + \gamma * \max_{a'} Q(o_{j+1}, a'; \theta) & \text{else.} \end{cases} \quad (4)$$
- $$\mathit{TaskLoss} = (y_j - Q(o_j, a_j; \theta))^2 \quad (5)$$

The training procedure runs for around 15k gradient descent steps and 3k environment steps. The training took between 20 minutes and 1 hour on a GeForce GTX 1080 Ti GPU. We used a learning rate of 5e-3 that decays to 5e-4 and then to 5e-5 after respectively 1000 and 10000 steps. Note that the hyperparameters used in the training were not optimized and we mainly aimed at solving the task, not necessarily in an efficient way.

Model initialization

We found that when trained directly for the task, the model was trapped in a local minimum where it outputted constant values, no matter the state of the inputs. We think that this is because there need to be iterated applications of the update rule on each of the intermediate cells between the inputs and outputs to transmit and modify the information. This repeated use of a neural network makes the gradient vanish, as observed in vanilla Recurrent Neural Networks [34].

To solve this problem, we first trained the neural CA on an easier task: both outputs were optimized to compute the mean of the inputs. We found that it was able to learn with a reasonably low error after several thousand gradient descent steps.

This initialization enables the neural CA to learn to stabilize the states of the cells, make an information link from input to output, and a linear combination of the input values at the output cells. This procedure is similar to what is used in curriculum learning [35] where an easy subset of the task is learned before tackling the whole problem. Here we did not use a sub-task as a starting point but a different task that shared common requirements. This initialization phase appears essential in the experiments we conducted. However, we do not think that this is specific to the task to be learned. This can be seen as a general "warm-up" phase to

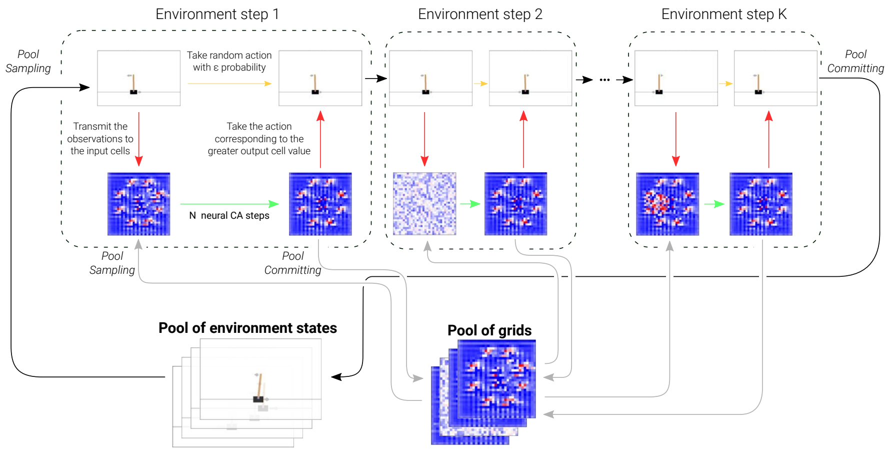

Environment step 1
Environment step 2
Environment step K
Pool Sampling
Take random action with ε probability
Pool Committing
Transmit the observations to the input cells
Take the action corresponding to the greater output cell value
N neural CA steps
Pool Sampling
Pool Committing
Pool of environment states
Pool of grids

Figure 6. The procedure for the exploration phase. We match cart-pole states and grids randomly sampled from two independent pools. This provides a way to simulate long-term dynamics both for the cart-pole environment and for the neural CA. In practice, we used K=2.

learn how to connect input and output cells.

The whole training procedure can be reproduced online in a Google Colab notebook[3].

Results

From estimating Q-values to stable behavior

To get a persistent control of the cart-pole agent, we begin by transmitting the observation of the current state in the input cells. We let the neural CA evolve for a random number of steps between 50 and 60. We take the action corresponding to the maximum output value and we input the new observation to the neural CA. The grid is initialized with uniform random values and the same grid is used during the whole simulation. After training, the cart-pole controller with neural CA is tested for how long the pole can remain balanced. Moreover, we verify its resistance to damage, noise, and input deletion.

Our model was able to solve the cart-pole problem and achieve long-term stability of both the pole balancing and of the states of the CA. It was able to balance the pole for more than 10k simulation steps. Detailed performance evaluation can be found in table 1.

Resulting neural CA

Beyond performance, it is interesting to visualize the activities of the neural CA during the control of the cart-pole agent.

In figure 8, we observe that the first 50 steps lead to a precise spatial organization of the grid and, in figure 7, this phase corresponds to the stabilization of the output values. Once stabilized, this global shape will not change during the whole run. This can be thought as the developmental part of the neural CA. This phase can be seen in the videos on the interactive version of this preprint available at https://avariengien.github.io/self-organized-control/.

3 https://colab.research.google.com/github/aVariengien/self-organized-control/blob/main/code/Towards-self-organized-control-notebook.ipynb

Then, during the remaining part of the simulation, the spatial activity is changing in phase with the physical observations, as shown in figure 9. This is the computing phase. Even if the two phrases seem to exist in the different models we found, the exact organization of the grid differs significantly, as visible in figure 10. It is exciting to see that a wide variety of shapes emerges from optimizing for the same function!

Note that the output values are always really close to one another. Since the pole is in a balanced state, the difference between the expected reward after going left or right is small. Going left then right or going right then left will not yield a great difference in total reward.

Robustness abilities

During training, the neural CA has always at least 50 steps between the update of the inputs, where damage can occur, and the readout of the output values. During testing, we also experimented with a more challenging type of damage we called *uniformly distributed damage* where at each CA step the grid has a constant probability of being damaged.

Because this type of damage was more difficult to cope with, we decreased the damage frequency: on average, the CA receives one damage every 4 input updates with uniformly distributed damage and one every 2 input updates with the damage used during training.

The performance of the neural CA with different perturbations is summarized in the table 1. The score denotes the number of environment steps before the pole falls, or the cart hits a wall. In each situation, we computed the mean score on 100 independent runs, as well as the standard deviation. To ease the analysis, we conducted the experiments of this section only on a single model, nonetheless, the main conclusions generalize to the other models.

Resistance to damage

We found that the neural CA was able to maintain its shape and its function despite frequent damage. In the figure 11 we can observe

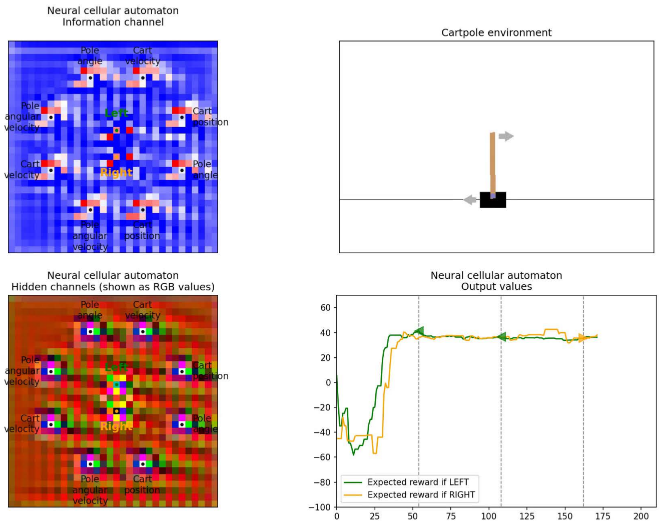


Figure 7. On the left, the states of the neural CA. For the information channel, negative values are in blue, positive in red while the hidden channel is represented as RGB values. Bottom right: the plot of the output values of the neural CA. The vertical dotted lines denote when the action that has the maximum expected reward is taken by the cart-pole agent. Green triangle: "push left" action, orange triangle: "push right" action. Videos of the cart-pole agent and the neural CA in action are available at https://avariengien.github.io/self-organized-control/

.

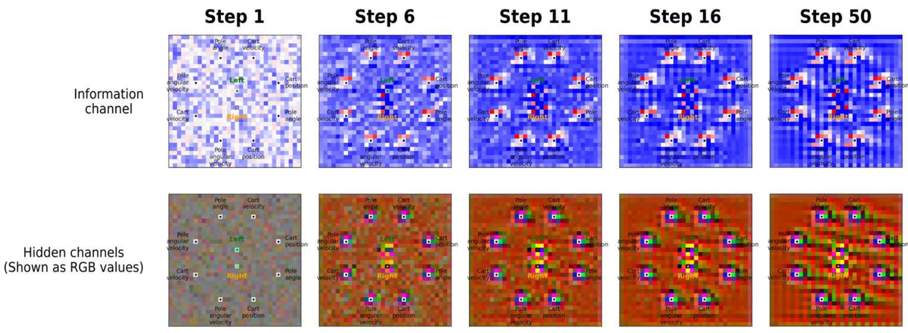


Figure 8. The state of the grid during the 50 first phase. The information channel is displayed on the top. Negative values are in blue, positive in red while the hidden channels (on the bottom) are represented as RGB values. We observe an evolution of the grid from a disorganized state to a precise, stable pattern.

	No damage	Damage after input update	Uniformly distributed damage
No noisy update	13273 ± 11905	2598.19 ± 2241	391.6 ± 283
With noisy update	1296.3 ± 899	739.7 ± 473	345.7 ± 214

Table 1. Performance of the cart-pole agent with several types of perturbation. The score is the number of time steps the cart-pole stays balanced without hitting walls. The scores are averaged on 100 runs and are noted *mean ± standard deviation*.

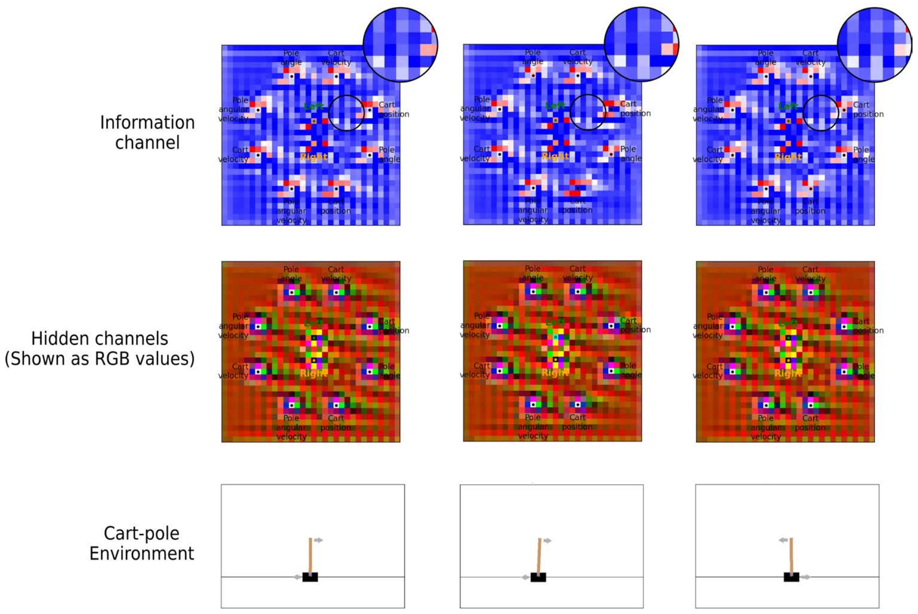


Figure 9. The state of the grid and of the environment in 3 situations, taken from the same run. The information channel is displayed on the top. Negative values are in blue, positive in red, while the hidden channels (on the bottom) are represented as RGB values. We observe that the global pattern of the grid remains highly similar. However, subtle variations around input cells are visible, as the ones highlighted in the circular enlargements of a grid region, on the top right of the grids.

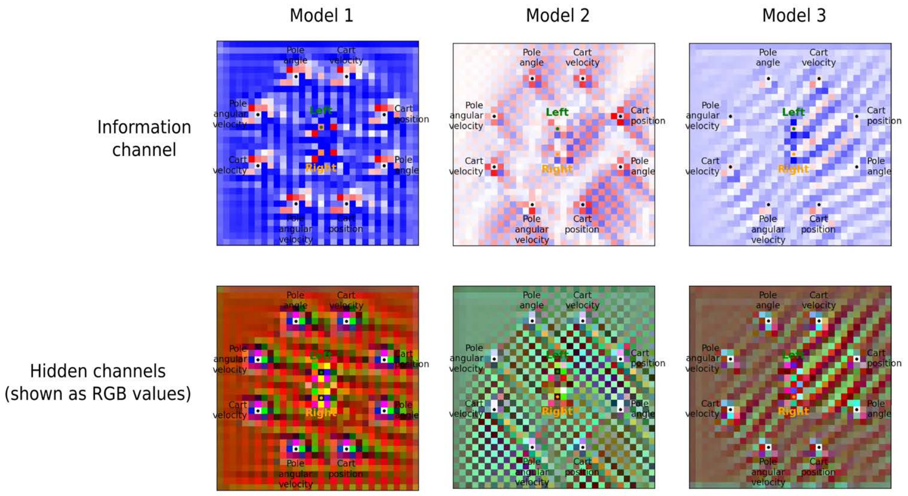


Figure 10. The stable spatial organization of the grid for 3 independently trained neural CA. The information channel is displayed on the top. Negative values are in blue, positive in red, while the hidden channels (on the bottom) are represented as RGB values. Their dynamics can be observed in videos available at https://avariengien.github.io/self-organized-control/

.

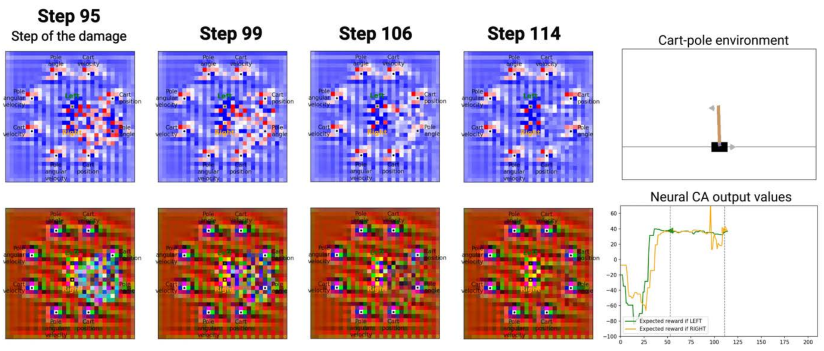


Figure 11. A slide sequence of grids during the recovery from damage. The information channel is displayed on the top. Negative values are in blue, positive in red, while the hidden channels (on the bottom) are represented as RGB values. On the left is the state of the cart-pole during the damage and the evolution of the output values of the neural CA. We can observe that the grid recovers its structure. Moreover, a temporary perturbation of the output value can be observed on the plot on the bottom left. Videos of the recovery can be found at https://avariengien.github.io/self-organized-control/

how the grid recovers its shape after damage. Although damage can lead to great perturbations in the output values and so to random actions, the agent is still able to stabilize the pole for several hundred steps.

Moreover, the neural CA was not trained to recover from uniformly distributed damage, this explains the greater diminution in the average score visible in the table 1.

Resistance to noise

The amount of noise added to each update is often of the same order as the difference between the two outputs when the pole is in a balanced state. This is why we observe in the figure 12 the green and orange curves are subject to stochastic variations that lead them to cross many times between each readout. The policy that controls the cart-pole agent is thus heavily randomized. Despite the noisy update, the neural CA can produce a probability distribution of actions such that a stable behavior emerges.

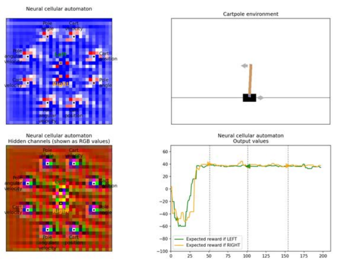


Figure 12. A neural CA controls a cart-pole agent with noisy updates. On the left is the grid of the neural CA. For the information channel, negative values are in blue, positive in red while the hidden channel is represented as RGB values. Bottom right: the plot of the output values of the neural CA. The vertical dotted lines denote when the action that has the maximum expected reward is taken by the cart-pole agent. Green triangle: "push left" action, orange triangle: "push right" action. We can observe more crossing of the plots of the output values due to the noisy update. The neural CA has to compensate for the random perturbation of the output values.

Resistance to input deletion

One of the particularities of this neural CA model is its flexibility. For instance, the number of inputs and outputs can vary without changing the architecture. We only have to replace input or output cells with intermediate ones.

We were interested in exploring this flexibility and whether our model showed robustness to input deletion. Because observations from the cart-pole environment are encoded redundantly, we tested if it was able to exploit this particularity even if it was not trained for this.

In the table 2, we can observe the consequences of deleting each input. We computed the mean scores on 25 independent runs for each input deletion. Each column corresponds to one observation type, each row corresponds to the top or the bottom input cell encoding this observation being deleted (see figure 3 for the position of the input cells).

The system seems to be dependent on a few input cells that seriously impair performances such as the top input cell encoding for pole angle and the ones corresponding to the angular velocity, while others seem not to significantly affect its abilities. We hypothesize that even if it has not been directly trained to be robust to input deletion, the robustness to damage and noise includes also adaptation to unseen perturbation.

It seems that the inputs corresponding to the cart position do not disturb the control abilities. So we experimented with how the system will react to sensory deprivation by removing these two input cells such that the system has no longer access to this observation. It is still able to maintain the pole balanced for several thousand steps (score of 3926.7 ± 2383 on 25 runs without noise and damage). The reconfiguration of the grid can be observed in the figure 13.

Influence field visualization

In the videos showing neural CA and the environment side to side, we can observe that the regions around input cells are producing a dynamic pattern in phase with the movements of the cart-pole.

	Cart position	Cart velocity	Pole angle	Pole angular velocity
Top input deleted	814.1 ± 632	293.4 ± 144	53.6 ± 37	103.2 ± 30
Bottom input deleted	814.6 ± 608	168.2 ± 46	924.6 ± 601	267.2 ± 139

Table 2. Mean score and standard deviation of 25 independent runs after each input cell deletion. The CA were perturbed with damage after the update (on average one every 2 cart-pole steps) and noisy update.

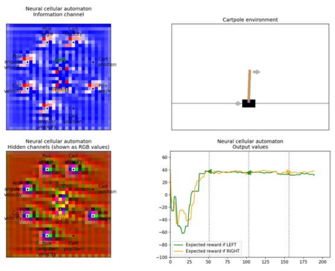


Figure 13. A neural CA controls a cart-pole agent with two deleted input cells. On the left, the neural CA grid. The input cells corresponding to the cart position have been deleted and replaced by intermediate cells. No noise nor damage was added. On the left is the grid of the neural CA. For the information channel, negative values are in blue, positive in red while the hidden channel is represented as RGB values. Bottom right: the plot of the output values of the neural CA. The vertical dotted lines denote when the action that has the maximum expected reward is taken by the cart-pole agent. Green triangle: "push left" action, orange triangle: "push right" action.

We develop a visualization tool to investigate the region of neural CA that is influenced by a particular input. To this end, we compared the evolution of the neural CA between a baseline case and a case where a particular input was perturbed. We then computed the relative mean of the difference for each of the cells in the grid, according to the formula (6). This process is repeated on different observations sampled from the environment and for several grids to get consistent patterns.

The expression of the deviation used to quantify the influence of a given input on the other cells is given in equation (6). The norm is the L2 norm and is computed by treating each cell as a 6-dimensional vector. In practice, the mean was computed for 50 different observations, and for each observation, we used 4 independent grids.

$$Deviation = \frac{\text{Mean}(\text{Norm}(Grid_{baseline} - Grid_{perturbed}))}{\text{Mean}(\text{Norm}(Grid_{baseline}), \text{Norm}(Grid_{perturbed}))} \quad (6)$$

We used as a perturbation the multiplication by a random number between -1 and 1. This ensures that the input will not be out of the range of the possible values while allowing for a sufficient range to get interpretable visualization. We experimented with different types of perturbation, the resulting visualizations were similar. Each input cell is perturbed independently: its sister input cell transmitting the same observation is not affected by the perturbation. The region of influence for each cell is visualized in the figure 14 for 3 different models.

For the model 1, we can observe a localized influence of the input cell with a tendency to be directed toward the right. We also discovered that the inputs that cause the least performance loss if deleted (see table 2) were the ones positioned on the right. We hypothesize that the input cells on the right side of the grid had less influence on the outputs because the CA learned a rule that can be summarized as "propagate information toward the right". This is allowed by the fact that information is redundant and that the majority of input cells on the right hold the same observation as an input cell on the left side. Because this propagation property seems also shared with the models 2, 3 and other models we trained, we hypothesize that the input cells placed on the left could hold the most useful combinations of information. Nonetheless, further experiments are needed to characterize a global direction of propagation depending on the position and on the nature of the input cells.

For the model 2, we observe a great amount of deviation even without any perturbation. This makes it difficult to interpret the results with perturbation. It seems that the influence of a given input cannot be visible by the fact that the values of another cell are affected, but in the way these values change.

The model 3 could be the intermediate between the two precedents. It presents more deviation without perturbation, while still exhibiting a clear increase in deviation localized around perturbed inputs.

The conclusions that can be drawn from these visualizations are still limited and must be taken carefully. This technique is shared as an attempt to understand the underlying dynamics of the resulting self-organizing system. We think that the development of visualization tools could be a useful step to direct the future design of self-organizing systems.

Discussion

In this work, we demonstrated that neural CA can be used as a differentiable black-box function theoretically extending its applications to the approximation of any functions. Here we demonstrated its abilities in the context of Deep-Q learning. We used it to solve the simple cart-pole problem. A direct future challenge would be to apply it to more challenging tasks where the input and output dimensionality is much higher.

The computing abilities of the neural CA were maintained over several hundreds of thousand iterations, producing an emergent stable behavior in the environment it controls for thousands of steps. Moreover, the system obtained demonstrated life-like phenomena such as a developmental phase, regeneration after damage, stability despite a noisy environment, and robustness to unseen disruption such as input deletion. In the future, we could also experiment with randomized input and output positions. This would add new challenges: recognize the role of each input and output cell and then create flexible pathways to transmit and combine information.

Even if the developmental phase and the computing phase used the same rules, our system cannot adapt to new environments once the training ends. Future works could explore the possibility of adding plasticity abilities and useful memory of past events stored in the states of the cells. This would mean that the neural CA could recognize a particular situation, and adapt its computations accordingly. Moreover, we envision that even metaplasticity found in biological neurons [36] could be achieved by neural CA.

Besides the biological plausibility of neural CA, their interest also relies on the fact that they are a highly decentralized computing model. Neural CA could be executed efficiently on dedicated hardware using locally connected microprocessors such as cellular neural networks [37]. Other works explored exciting directions such as framing reaction-diffusion mechanisms as neural CA [31]

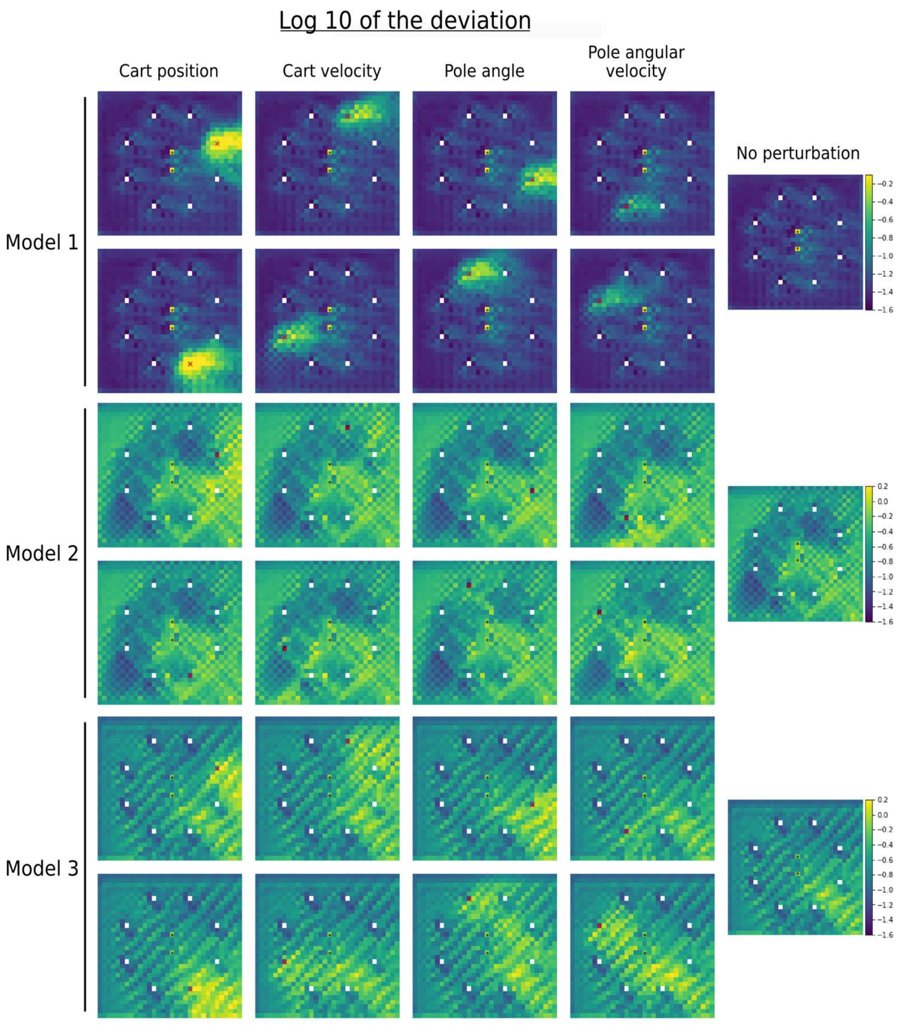


Figure 14. Visualization of the region of influence of each input cell. We plotted the decimal logarithm of the deviation between a standard input and a perturbed one. The perturbed input cell is symbolized by a red cross, black dots identify the output cells. The natural deviation without any perturbation, due to the stochasticity of the system is shown on the right.

that would potentially lead to implementation using chemical computing. Those reaction-diffusion systems could also be applied to other tasks than shape homeostasis, such as control.

Beyond the dissociation of the environment and the controller, we could imagine a neural CA that could perform both shape homeostasis and controls the movement of this shape at the same time. If a suitable physical implementation is found, such works could give rise to new robotics and artificial devices with self-organizing abilities that are for now reserved for the living world.

Additional experiments

In addition to the cart-pole balancing problem, we explored other tasks and different variations of the neural CA model. Here is a short list of the other tasks we tried that relate to problems solved by biological organisms. To keep this paper short, we chose to focus on a single task, but videos of our additional results and the code to reproduce them can be found at https://github.com/aVariengien/self-organized-control/tree/main/code/AdditionalExperiments.

- Exploring an environment to find a target cell: In this task, new cells can grow only next to already living cells. Each cell has an energy value that controls its fire rate. The goal is, starting from a single alive cell, to find a randomly placed target while using in total the lowest amount of energy.
- Following a gradient: This task is similar to the previous but we provide information for the position of the output. Each cell possesses a read-only channel that is proportional to the distance to the target. This way, the growth can be directed toward the target cell instead of being limited to strategies of random exploration.
- Computing Boolean functions: We experimented with computing simple Boolean functions such as XOR or its negation, NOT XOR. The environment includes 2 input and 1 output cells. In addition to damage and noise, the position of the input and output cells are randomized such that to solve the task, the cells must communicate without relying on fixed positions.

Acknowledgment

This work was partially funded by the Norwegian Research Council (NFR) through their IKTPLUSS research and innovation action under the project Socrates (grant agreement 270961), and Young Research Talent program under the project DeepCA (grant agreement 286558).

Author contributions

SN and SPF have discussed the initial idea for the work. AV has conducted the experimental work. SN, SPF, and TG have supervised the work. All authors have participated in brainstorming sessions throughout the work. AV has drafted the manuscript. SN, SPF and TG have reviewed the manuscript. SN has secured funding for the work.

References

1. Sansom SN, Livesey FJ. Gradients in the brain: the control of the development of form and function in the cerebral cortex. Cold Spring Harbor perspectives in biology 2009;1(2):a002519.
2. Ward NS. Neural plasticity and recovery of function. Progress in brain research 2005;150:527–535.
3. Gougoux F, Zatorre RJ, Lassonde M, Voss P, Lepore F. A functional neuroimaging study of sound localization: visual cortex activity predicts performance in early-blind individuals. Plos biol 2005;3(2):e27.
4. Levin M. Bioelectromagnetics in morphogenesis. Bioelectromagnetics 2003;24(5):295–315.
5. Zador AM. A critique of pure learning and what artificial neural networks can learn from animal brains. Nature communications 2019;10(1):1–7.
6. Banzhaf W, Miller J. The challenge of complexity. In: Frontiers of Evolutionary Computation Springer; 2004.p. 243–260.
7. Stanley KO, Miikkulainen R. Efficient evolution of neural network topologies. In: Proceedings of the 2002 Congress on Evolutionary Computation. CEC'02 (Cat. No. 02TH8600), vol. 2 IEEE; 2002. p. 1757–1762.
8. Nadizar G, Medvet E, Pellegrino FA, Zullich M, Nichele S. On the effects of pruning on evolved neural controllers for soft robots. In: Proceedings of the Genetic and Evolutionary Computation Conference Companion; 2021. p. 1744–1752.
9. Miller JF. Evolving developmental neural networks to solve multiple problems. In: Artificial Life Conference Proceedings MIT Press; 2020. p. 473–482.
10. Mixter J, Akoglu A. Growing Artificial Neural Networks. arXiv preprint arXiv:200606629 2020;.
11. Nichele S, Ose MB, Risi S, Tufte G. CA-NEAT: evolved compositional pattern producing networks for cellular automata morphogenesis and replication. IEEE Transactions on Cognitive and Developmental Systems 2017;10(3):687–700.
12. Mordvintsev A, Randazzo E, Niklasson E, Levin M. Growing neural cellular automata. Distill 2020;5(2):e23.
13. Gregor K, Besse F. Self-Organizing Intelligent Matter: A blueprint for an AI generating algorithm. arXiv preprint arXiv:210107627 2021;.
14. Chan BWC. Lenia and expanded universe. In: Artificial Life Conference Proceedings MIT Press; 2020. p. 221–229.
15. Diedrich FJ, Warren Jr WH. Why change gaits? Dynamics of the walk-run transition. Journal of Experimental Psychology: Human Perception and Performance 1995;21(1):183.
16. Taga G, Yamaguchi Y, Shimizu H. Self-organized control of bipedal locomotion by neural oscillators in unpredictable environment. Biological cybernetics 1991;65(3):147–159.
17. Amari Si. Dynamics of pattern formation in lateral-inhibition type neural fields. Biological cybernetics 1977;27(2):77–87.
18. Bicho E, Louro L, Erlhagen W. Integrating verbal and nonverbal communication in a dynamic neural field architecture for human-robot interaction. Frontiers in neurorobotics 2010;4:5.
19. Carver CS, Scheier MF. Control processes and self-organization as complementary principles underlying behavior. Personality and social psychology review 2002;6(4):304–315.
20. Neumann J, Burks AW. Theory of self-reproducing automata, vol. 1102024. University of Illinois press Urbana; 1966.
21. Gerlee P, Basanta D, Anderson AR. The influence of cellular characteristics on the evolution of shape homeostasis. Artificial life 2017;23(3):424–448.
22. Miller JF. Evolving a self-repairing, self-regulating, french flag organism. In: Genetic and Evolutionary Computation Conference Springer; 2004. p. 129–139.
23. Bidlo M, Škarvada J. Instruction-based development: From evolution to generic structures of digital circuits. International Journal of Knowledge-Based and Intelligent Engineering Systems 2008;12(3):221–236.
24. Bidlo M. On routine evolution of complex cellular automata. IEEE Transactions on Evolutionary Computation 2016;20(5):742–754.
25. Mitchell M, Hraber P, Crutchfield JP. Revisiting the edge of chaos: Evolving cellular automata to perform computations. arXiv preprint adap-org/9303003 1993;.
26. Nichele S, Tufte G. Evolutionary growth of genomes for the development and replication of multicellular organisms with indirect encoding. In: 2014 IEEE International Conference on Evolvable Systems IEEE; 2014. p. 141–148.
27. Nichele S, Glover TE, Tufte G. Genotype regulation by

self-modifying instruction-based development on cellular automata. In: International Conference on Parallel Problem Solving from Nature Springer; 2016. p. 14–25.

28. Sudhakaran S, Grbic D, Li S, Katona A, Najarro E, Glanois C, et al. Growing 3D Artefacts and Functional Machines with Neural Cellular Automata. arXiv preprint arXiv:210308737 2021;.
29. Horibe K, Walker K, Risi S. Regenerating Soft Robots Through Neural Cellular Automata. In: EuroGP; 2021. p. 36–50.
30. Randazzo E, Mordvintsev A, Niklasson E, Levin M, Greydanus S. Self-classifying MNIST Digits. Distill 2020;5(8):e00027–002.
31. Mordvintsev A, Randazzo E, Niklasson E. Differentiable Programming of Reaction-Diffusion Patterns. arXiv preprint arXiv:210706862 2021;.
32. Kingma DP, Ba J. Adam: A method for stochastic optimization. arXiv preprint arXiv:14126980 2014;.
33. Mnih V, Kavukcuoglu K, Silver D, Graves A, Antonoglou I, Wierstra D, et al. Playing atari with deep reinforcement learning. arXiv preprint arXiv:13125602 2013;.
34. Hochreiter S. The vanishing gradient problem during learning recurrent neural nets and problem solutions. International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems 1998;6(02):107–116.
35. Bengio Y, Louradour J, Collobert R, Weston J. Curriculum learning. In: Proceedings of the 26th annual international conference on machine learning; 2009. p. 41–48.
36. Abraham WC. Metaplasticity: tuning synapses and networks for plasticity. Nature Reviews Neuroscience 2008;9(5):387–387.
37. Cimagalli V, Balsi M, Caianiello E. Cellular neural networks: A review. In: Neural Nets WIRN Vietri'93: Proc. of 6th Italian Workshop (Salerno, 1993) World Scientific; 1993. p. 55–84.

Algorithm 1 The exploration procedure to gather experiences of the environment and store them in the memory of the agent. NCA is the neural CA function that updates the grids, G is the pool of grids, S the pool of states, D the agent memory and K the number of environment steps to explore.

```
procedure EXPLORE(NCA, $G$, $S$, $D$, $K$)
    Sample $s_1$ from $S$
    Set the environment $E$ to state $s_1$ and observe $o_1$
    for $t$=1 to $K$ do
        Sample $GridBatch$ from $G$
        $N \leftarrow$ RandomInt($Step_{min}$, $Step_{max}$)
        for $step$ = 1 to $N$ do
            $GridBatch \leftarrow$ NCA($GridBatch$, $o_t$)    ▷ NCA compute a neural CA step on a batch of grids.
        end for
        for $grid$ in GridBatch do
            With probability $d$, perform damage on $grid$
        end for
        Commit $GridBatch$ back to $G$
        Sample $Grid$ from $GridBatch$
        With probability $\epsilon$ select a random action $a_t$
        Otherwise select $a_t = \text{argmax}_a Grid_{x_a, y_a, 0}$
        Execute $a_t$ in $E$ and observe reward $r_t$ and observation $o_{t+1}$
        Store the transition $(o_t, a_t, r_t, o_t + 1)$ in $D$
        if t+1 is terminal then
            Set $E$ to an initial state and observe $o_{t+1}$
        end if
    end for
end procedure
```

Algorithm 2 The implementation of the deep-Q learning algorithm applied to neural CA. NCA is the neural CA function that updates the grids, G is the pool of grids, D is the agent memory and R is the number of transitions to sample in the agent memory to train the NCA.

```
procedure TRAIN(NCA, $G$, $D$, $R$)
    Sample $R$ transitions $(o_{t_j}, a_{t_j}, r_{t_j}, o_{t_j+1})$ from $D$ of previous time steps $t_0, ..., t_{R-1}$
    for $i$ = 0 to $\frac{R}{B} - 1$ do
        Arrange the transitions of time steps $t_{i*B}, t_{i*B+1}, ..., t_{(i+1)*B-1}$ in a batch $T_i$
        Sample $GridBatch_i$,$GridBatch'_i$ from G
        Match each transition $(o_{t_k}, a_{t_k}, r_{t_k}, o_{t_k+1})$ from $T_i$ to a $Grid_k$ in $GridBatch_i$ and to a $Grid'_k$ in $GridBatch'_i$
        for k=1 to B do
            for $step$ = 1, RandomInt($Step_{min}$, $Step_{max}$) do
                $Grid_k \leftarrow$ NCA($Grid_k$, $o_{t_k+1}$)    ▷ To ease the reading of the pseudocode, NCA can also be applied to individual grids.
            end for
            $y_k \leftarrow \begin{cases} r_{t_k} & \text{if } t_k \text{ is a final step} \\ r_{t_k} + \gamma * \text{max}_a Grid_{k, x_a, y_a, 0} & \text{else.} \end{cases}$
            for $step$ = 1, RandomInt($Step_{min}$, $Step_{max}$) do
                $Grid'_k \leftarrow$ NCA($Grid'_k$, $o_{t_k}$)
            end for
            $TaskLoss_k \leftarrow (Grid'_{k, x_{a_{t_k}}, y_{a_{t_k}}} - y_k)^2$
            $Loss_k \leftarrow TaskLoss_k + \lambda *$ OverFlow($Grid'_k$)
        end for
        Compute the gradient $Grad$ of $\sum_{k=1}^{B} Loss_k$ with respect to the parameters of NCA using BPTT
        Update the parameters of NCA using $Grad$ and a gradient-descent based optimization
    end for
end procedure
```

Algorithm 3 The full procedure to train a neural CA to perform a reinforcement learning task.

```
Initialize the agent memory $D$ empty
Initialize randomly the pool of grids G with $M$ batches of $B$ grids
Initialize the pool of environment state S with $L$ initial states
Initialize NCA
for iteration=1 to $I$ do
    EXPLORE(NCA,G,$S$, $D$,$K$)
    TRAIN(NCA,G, $D$)
end for
```

Alexandre Variengien is a master student in computer science at the École Polytechnique Fédérale de Lausanne (EPFL, Lausanne, Switzerland) in a dual degree program with the École Normale Supérieure de Lyon (ENS de Lyon, Lyon, France) with a strong interest for research. His interest focuses on the phenomenon of emergence in both artificial and biological systems. He is particularly interested in the development of interdisciplinary initiatives applied to artificial intelligence.

Sidney Pontes-Filho is a computer scientist who specialized in Artificial Intelligence. He is currently a Ph.D. candidate in Computer Science at the Norwegian University of Science and Technology (NTNU, Trondheim, Norway) and works as a Ph.D. fellow at Oslo Metropolitan University (OsloMet, Oslo; Norway) since 2018. He received his B.Sc. in Computer Science in 2013 from Federal University of Paraíba, Brazil, and M.Sc. in Computer Science in 2018 from Technical University of Kaiserslautern, Germany. His current research interests are complex systems, unconventional computing, computational neuroscience, and artificial general intelligence.

Tom Eivind Glover is a computer scientist specializing in artificial intelligence. He is currently a Ph.D. candidate in engineering science at Oslo Metropolitan University (OsloMet, Oslo, Norway) since 2020. He received his B.Sc. in computer science in 2014 and a M.Sc. in computer science in 2016, both from the Norwegian University of Science and Technology (NTNU, Trondheim, Norway). Prior to his PhD, he worked as an IT consultant/programmer for 4 years. Current research interests include reservoir computing, cellular automata, unconventional computing, and complex systems.

Stefano Nichele is a full professor of data science at the Oslo Metropolitan University (Oslo, Norway) and an adjunct research scientist at the Simula Metropolitan Centre for Digital Engineering (Oslo, Norway). He is a senior IEEE member. His research interests include artificial life, complex systems, cellular automata, and evolutionary-developmental systems.

Paper C.2

A unified substrate for body-brain co-evolution
(Pontes-Filho et al., 2022)

Author(s):
Sidney Pontes-Filho, Kathryn Walker, Elias Najarro, Stefano Nichele and Sebastian Risi

Published at workshop:
From Cells to Societies: Collective Learning Across Scales at Tenth International Conference on Learning Representations (ICLR 2022)

Note:

This work was also published as a poster paper titled "A single neural cellular automaton for body-brain co-evolution" at The Genetic and Evolutionary Computation Conference (GECCO 2022) with copyright © 2022 ACM, Inc.

Paper C.2
Pontes-Filho et al. (2022)

A UNIFIED SUBSTRATE FOR BODY-BRAIN CO-EVOLUTION

Sidney Pontes-Filho[1,2,*], **Kathryn Walker**[3], **Elias Najarro**[3], **Stefano Nichele**[1,4,5] **and Sebastian Risi**[3]

[1]Department of Computer Science, Oslo Metropolitan University, Oslo (Norway)
[2]Department of Computer Science, Norwegian University of Science and Technology, Trondheim (Norway)
[3]Digital Design Department, IT University of Copenhagen, Copenhagen (Denmark)
[4]Department of Holistic Systems, Simula Metropolitan Centre for Digital Engineering, Oslo (Norway)
[5]Department of Computer Science and Communication, Østfold University College, Halden (Norway)
[*]Corresponding author: sidneyp@oslomet.no

ABSTRACT

The discovery of complex multicellular organism development took millions of years of evolution. The genome of such a multicellular organism guides the development of its body from a single cell, including its control system. Our goal is to imitate this natural process using a single neural cellular automaton (NCA) as a genome for modular robotic agents. In the introduced approach, called *Neural Cellular Robot Substrate* (NCRS), a single NCA guides the growth of a robot and the cellular activity which controls the robot during deployment. In this paper, NCRSs are trained with covariance matrix adaptation evolution strategy (CMA-ES), and covariance matrix adaptation MAP-Elites (CMA-ME) for quality diversity, which we show leads to more diverse robot morphologies with higher fitness scores. While the NCRS can solve the easier tasks from our benchmark environments, the success rate reduces when the difficulty of the task increases. We discuss directions for future work that may facilitate the use of the NCRS approach for more complex domains.

1 INTRODUCTION

Multicellular organisms are made of cells that can divide into many, which specialize in controlling and maintaining the body, sensing the environment, or protecting from external threats. Such features were acquired by evolution from the first living cell. After millions of years, colonies of unicellular organisms appeared and were essential to the development of multicellular organisms with cellular differentiation (Niklas & Newman, 2013). Developmental biologists study that the growth and specialization of an organism are coordinated by its genetic code (Slack & Dale, 2021).

The field of artificial life tries to create life-like computational models taking ideas from biological life, such as decentralized and local control (Langton, 2019). One of the sub-fields of artificial life, artificial development (Harding & Banzhaf, 2009; Doursat et al., 2013), focuses on modeling or simulating cell division and differentiation. The techniques applied in artificial development are often based on the indirect encoding of developmental rules (i.e. analogous to the genome of a biological organism describing its phenotype). This type of encoding facilitates the scaling of an organism because the information in the genome is much smaller than in the resulting phenotype. This property is referred to as genomic bottleneck (Zador, 2019; Variengien et al., 2021), and it implies that the genetic code of an organism compresses the information to grow and maintain its body, and in some species even complex brains.

One of the simplest computational models of artificial life or dynamical systems is a cellular automaton (CA) (Wolfram, 2002). A CA can be described as a universe with discrete space and time, which is governed by local rules without any central control. Such a discrete space is divided into a regular grid of cells and can possess any number of dimensions. The most commonly studied CAs have one or two dimensions and their most well-known versions are, respectively, elementary CA (Wolfram, 2002) and Conway's Game of Life (Conway et al., 1970). Both have cells with binary states, but other CA can have many discrete states or continuous ones. In the 1940s, the first CA was introduced by Ulam and von Neumann (Topa, 2011). Von Neumann aimed to produce self-replicating machines, and Ulam worked on crystal growth. In 2002, a CA with rules defined by an artificial neural network was described (Li & Yeh, 2002). Nowadays, this type of approach is called neural cellular automaton (NCA). In 2017, Nichele et al. (2017) presented an NCA that has developmental features that were learned through neuroevolution using a method called compositional pattern-producing network (Stanley, 2007). Recently, Mordvintsev et al. (2020) introduced a differentiable NCA, which possesses growth and regeneration properties. In their work, an NCA is trained through gradient descent to grow a colored image from one active "seed" cell.

In evolutionary robotics, co-evolution of morphology and control has the inherent challenge of optimizing two different features in parallel (Bhatia et al., 2021). It also presents scalability issues when it deals with modular robots (Yim et al., 2007). Our goal is to implement an approach where the optimization happens in just one dynamical substrate with local interactions. Here we introduce such a system, a *Neural Cellular Robot Substrate* (NCRS), in which a single NCA grows the morphology of an agent's body and also controls how that agent interacts with its environment. The NCA has two phases (Fig. 1). First is the developmental phase, in which the robot's body is grown, including where to place its sensors and actuators. In the following control phase, copies of the same NCA are running in each cell of the agent, taking into account only local information from neighboring cells to determine their next state. The optimization task thus entails figuring out how to transmit information from the robot's sensors to its actuators to perform the task at hand.

We also introduce a virtual environment with three benchmark tasks for evaluating the NCRS' capacity of designing a robot and then controlling it. Two benchmarks consist in growing and controlling a robot to approach a light source (Fig. 1b and Fig. 2a). The third task challenges the robot to carry a ball to a target area. In this benchmark, a second type of rudimentary eye is added, so the robot can differentiate the ball and the target area (Fig. 2b).

The main contribution of this work is the introduction of a single neural cellular automaton that first grows an agent's body and then controls it during deployment. While the solved benchmark domains are relatively simple, the unified substrate for both body and brain opens up interesting future research directions, such as opportunities for open-ended evolution (Stanley, 2019). The source code of this project is available at `https://github.com/sidneyp/neural-cellular-robot-substrate`.

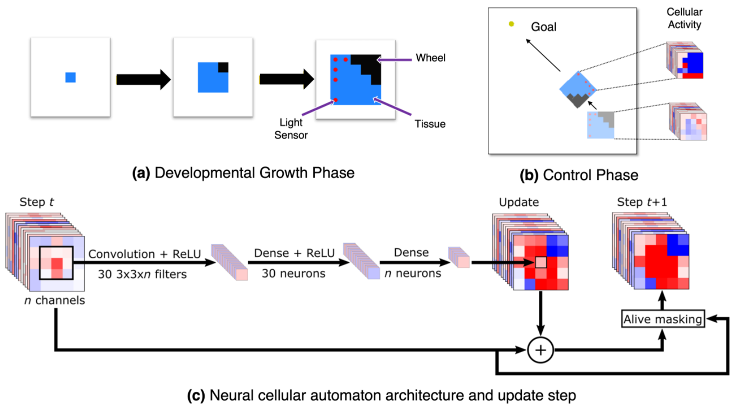
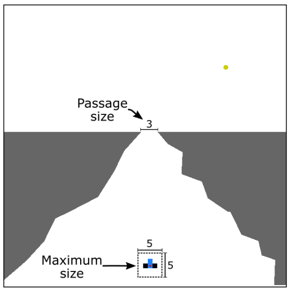
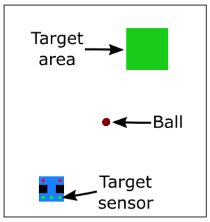


Figure 1: Neural Cellular Robot Substrate (NCRS). In the developmental phase (a), the robot is grown from an initial starting seed, guided by a neural cellular automaton (c). Once grown, the same neural cellular automaton determines how the signals propagate through the robot's morphology during the control phase (b).



(a) Light chasing with obstacle task



(b) Carrying ball to target task

Figure 2: Extensions from the light chasing task. (a) It depicts the original size of the playing field, which is 60.

2 Related work

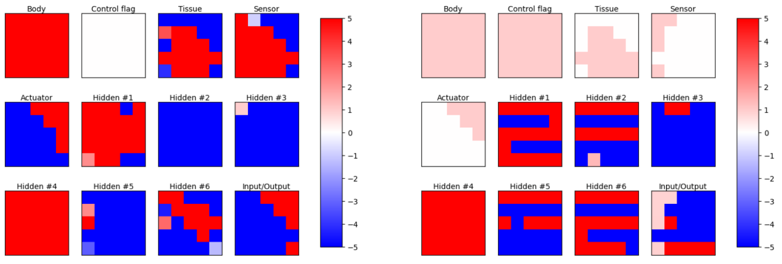
The co-design of robot bodies and brains has been an active area of research for decades (Medvet et al., 2021; Sims, 1994; Komosiński & Ulatowski, 1999; Veenstra & Glette, 2020; Gupta et al., 2021). Brain and body co-design stands for producing a control policy and a morphology for a robotic system. For example, in the work of Lipson & Pollack (2000) the same genome directly encodes the robot's body and the artificial neural network for control. A method that uses genetic regulatory networks to define separately a body and an artificial neural network was introduced by Bongard & Pfeifer (2003) and named artificial ontogeny. The evolved robots are able to locomote and push blocks in noisy environments. More recent work by Bhatia et al. (2021) presents several virtual environments and also an algorithm for brain and body co-design with separated description methods for the morphology and control. In comparison with NCRS, our co-design algorithm consists of only one neural cellular automaton.

The work on NCAs by Mordvintsev et al. (2020) is one of the first examples of self-organizing and self-repair systems that use differentiable models as rules for cellular automata. Before that, NCA models were typically optimized with genetic algorithms (Nichele et al., 2017). After the work on growing NCA, other neural CAs were introduced, including methods optimized without differentiable programming. There exist other generative methods for growing 3D artifacts and functional machines (Sudhakaran et al., 2021), for developing soft robots (Horibe et al., 2021). Moreover, an NCA was used as a decentralized classifier of handwritten digits (Randazzo et al., 2020).

The developmental phase of our approach is similar to the generative method with NCA for level design trained with CMA-ME in the work of Earle et al. (2021). Morphology design is also present in other works (Hejna III et al., 2021; Talamini et al., 2021; Kriegman et al., 2018; Brodbeck et al., 2015). The control phase is based on the NCA for controlling a cart-pole agent introduced by Variengien et al. (2021), but their NCA is trained using a reinforcement learning algorithm named deep-Q learning and the communication between NCA and environment happens in predefined cells. Our approach, NCRS, unifies these two methods by having two phases. The first phase is generative, and the second one is an agent's policy.

3 Approach: A Unified Substrate

The modular robots grown by the NCA consist of different types of cells such as sensors, actuators, and connecting tissue. After growth, the robot is deployed in its particular environment. Importantly, in our approach, the same NCA controls both the growth of the modular robot (Fig. 1a) and the robot itself (Fig. 1b). Therefore, it is a unified substrate for body-brain co-design and is called Neural Cellular Robot Substrate (NCRS). The architecture of NCRS is illustrated in Fig. 1c. When the



(a) Final time-step of developmental phase (b) Final time-step of control phase

Figure 3: Channels of the neural cellular automaton in different stages.

growth process is finished, the channels responsible to define the body modules reflect the robot's morphology, then the NCA can observe and act in the environment using the cells assigned to the specific types of modules, which are sensors, wheels, and tissue.

The state of a cell is updated considering the eight surrounding neighbors and itself, then it forms a 3×3 neighborhood. The values of the nine cells with all the n channels are processed by a trainable convolutional layer with 30 filters of size $3 \times 3 \times n$. Followed by a dense layer of 30 neurons and another one with n neurons for the n channels of the neural CA. After all cells have been computed, the result of this process is added to the previous state of the neural CA, and then it is clipped to the range of $[-5, 5]$. This update is only valid for the cells that are considered "alive", which are the ones that have their value in the body channel greater than 0.1 and their neighbors. This architecture is very similar to the ones in self-classifying MNIST (Randazzo et al., 2020) and in self-organized control of a cart-pole agent (Variengien et al., 2021).

The channels have specific roles in the neural CA, as shown in Fig. 3. The number of channels n differs because of the different number of sensors in the types of benchmark tasks. The body channel is the one that indicates that there is a body part in that cell if its value is greater than 0.1. The neighbors of a body part are allowed to update their states because they are considered "growing". The next channel has fixed values and works as a control flag. When the neural CA is in the developmental phase, all cells in this channel are set to zero. When it is in the control phase, they are set to one. The following channels are responsible to define the type of the body part. The channel with the highest value is the one that specifies the body part. In the case of a tie, the first channel is selected. The order of those channels is: body/brain tissue, light/ball sensor, target area sensor (if needed), and wheel. In this way, it can define a robot as depicted in Fig. 1a. Then, there are the hidden channels to support the computation in the neural CA. For all benchmark types, the neural CA contains six hidden channels. Finally, the input/output channel, which is the one that receives the values from the sensors and gives the values to the actuators (wheels).

The initial state of the neural CA is a "seed". The middle cell of the grid has the state set as one in the body channel, and the rest is zero. After a few time-steps in the developmental phase, all channels are updated except the control flag channel. This phase lasts for ten time-steps. The end of the developmental phase is represented in Fig. 3a. After development, the control phase starts. In this phase, the benchmark environment initializes with the developed robot body. For advancing one time-step in the environment, the NCA takes two time-steps for defining an action after receiving observations from the sensors. The body and body parts channels become fixed and their values are defined by the robot body. This is used to support the neural CA by identifying the cells with body modules, such as tissues, sensors, and actuators. The cell is assigned the value one to the body channel if there is a body part and to the specific body parts channel. Fig. 3b shows this assignment for the identification of body parts during the control phase. The robot designed by this NCA is depicted in Fig. 4a. At the start of the control phase, the cellular activity of the hidden and input/output channels is set to zero. In the input/output channel, only the input cells are fixed and their values come from the sensors.

In our neural CA, there is no noise. Even all "alive" cells are updated every time-step. This is done because the stochastic update or any other type of noise would affect the development of the robot body. After the developmental phase, the same model could produce different types of robot body.

For our experiments, the neural cellular automaton has a grid of size 5×5. Therefore, it generates a body for an agent with the same size. Since this is a neural CA, the grid size does not affect the number of trainable parameters. The light chasing and light chasing with obstacle environments require just the light sensor. Therefore, the robot can have tissue, light sensor and wheel. A wheel's orientation is always vertical during the initialization of the benchmark environment. The wheel rotates upwards and downwards relative to the initial angle of the robot. The maximum speeds for each of those directions are, respectively, +1 and -1. This takes three body part channels. With one body, one control flag, six hidden, and one input/output channels, the total number of channels is 12. In this way, the number of trainable parameters is 4,572. In the carrying ball to target environment, the robot needs one additional sensor. Therefore, it adds one more channel. It results in a neural CA with 4,873 trainable parameters.

4 BENCHMARK ENVIRONMENTS

To test the capacity of controlling the developed robot, we implemented three benchmark environments, which are: light chasing (LC), light chasing with obstacle (LCO), and carrying ball to target (CBT). They are environments where a modular robot equipped with simple light sensors and wheels can be evaluated. In those environments, we decided that the size of the playing field and the distance between the objects are affected by the maximum size that the robot can have. Thus, the larger the robot can be, the bigger the playing field. In our experiments, we use a robot and a neural cellular automaton grid with size 5×5. Because the possible maximum size of the robot is 5, we chose the size of the playing field to be 60.

The fitness score is calculated using the average score of 12 runs where the location of the agent, light, ball, and target can differ for each run. The light or ball has some predefined regions to be initially placed.

The benchmark environments are based on the implementation of the top-down racing environment in Open AI gym (Brockman et al., 2016). We use the pybox2d, which is a 2D physics library in Python.

4.1 LIGHT CHASING

The light chasing (LC) environment is shown in Fig. 1b. The goal of the agent is to be closer to the light during the entire simulation. The agent starts in the middle of the playing field. One light is randomly placed around the region of one of the four corners of the playing field. The fitness score is calculated by the average distance between the center of the robot and the center of the light over all simulation time-steps, and a successful run means that this distance reached less than 10 times the module size. The activity s of the agent's light sensors is calculated as:

$$s = e^{-distance/playfield}, \tag{1}$$

where the distance between the objects is normalized by the size of the playing field $playfield$, which is 60. The values of the sensor activity or fitness score are between 0 and 1, where 1 means no distance. The values exponentially decay to 0 with an increase in distance.

4.2 LIGHT CHASING WITH OBSTACLE

The light chasing with obstacle (LCO) environment is a more difficult version of the light chasing one (Fig. 2a). The robot does not have sensors to detect the obstacle, thus its morphology plays a bigger role in this benchmark. The passage width is calculated by the possible maximum size of the robot. If the robot can have up to 5×5 body parts, then the passage width would be the size of three body parts. The robot is randomly initialized at the bottom of the playing field. An obstacle is procedurally generated with a target passage width and wall roughness. The obstacle has the shape of a funnel because there are no sensors to it, then this helps the robot to reach the passage depending on its body. The passage is randomly located on the horizontal axis and fixed on the vertical axis.

The light is at the top and after the obstacle. The initial light location has four predefined regions on the horizontal axis, which are left, center-left, center-right and right. The fitness and success definition are the same as the light chasing task.

4.3 Carrying ball to target area

Among the three benchmark environments, the task to carry a ball to a target area is the most difficult one (Fig. 2b). For the control phase, the robot needs to move towards the ball, and then move to the target area without losing the ball during the transport. For the developmental phase, the body of the robot needs to be adequate to push or kick the ball to the target area, and properly placing the sensors of each type, so it can successfully locate ball and target area. The agent is located at the bottom in a random horizontal location. The ball is located in the middle of the vertical axis of the playing field, but it has the same four predefined regions as the light chasing with obstacle environment. The target is located at the top and its location on the horizontal axis is randomly defined. Besides the sensor for the ball (or light for the other two environments), there is a new sensor type that calculates the distance to the center of the target area (following equation 1).

The fitness score of this environment is the average of the distance between robot and ball, and the distance between the ball and the center of the target area. Since they are distances used to calculate the fitness score, they are normalized using equation 1. The definition of success in this task means carrying the ball to the target, so it can have a distance less than ten times the module size of the robot.

5 Training methods

We have chosen to use some derivative-free optimization methods because NCRS needs some adjustments for using deep reinforcement learning because of the variable number of inputs and outputs (Variengien et al., 2021). They are the covariance matrix adaptation evolution strategy (CMA-ES) (Hansen & Ostermeier, 1996) and covariance matrix adaptation MAP-Elites (CMA-ME) (Fontaine et al., 2020). The latter is used to add quality diversity to the former, broadening the exploration of robot designs. For both training methods, we use the library CMA-ES/pycma (Hansen et al., 2019). There are two training methods and three benchmark tasks. This gives a total of six different combinations. Because of the computational demands, each of these combinations was trained only once.

The training process is performed entirely on a CPU. To speed up evaluation times, robots with a design that would not work properly in the environment are not simulated. For the two light chasing environments, robots must have at least one light sensor and two actuators. For the carrying ball to target, they must have one sensor of each type and two actuators. The fitness scores of the failed designs are calculated according to the number of correct parts they have. For each correct body part, the fitness score increases by 0.01.

To compare the quality diversity of CMA-ES and CMA-ME, we use the percentage of cells or feature configurations filled, and QD-score. They measure quality and diversity of the elites (Pugh et al., 2016). The QD-score is calculated by summing the fitness score of all elites and dividing it by the total number of possible feature configurations. Moreover, CMA-ES and CMA-ME have their elites stored, even though CMA-ES does not use elites during training.

5.1 Covariance matrix adaptation evolution strategy

CMA-ES is one of the most effective derivative-free numerical optimization methods for continuous domains (Fontaine et al., 2020). CMA-ES runs 20,000 generations for all environments. The initial mean is 0.0 for all dimensions, and the initial coordinate-wise standard deviation (step size) is 0.01. The population size or the number of solutions acquired to update the covariance matrix is 112. This number was selected by the number of available threads in the machine used to train, which contains 56 threads at 2.70GHz.

Table 1: Best fitness score after training in the tasks of light chasing (LC), light chasing with obstacle (LCO) and carrying ball to target (CBT)

	CMA-ES	CMA-ME
LC	0.58274	0.61481
LCO	0.49295	0.47723
CBT	0.48445	0.47884

5.2 Covariance matrix adaptation MAP-Elites

CMA-ME is a variant of CMA-ES with the added benefit of quality diversity from MAP-Elites (Mouret & Clune, 2015). The changes to CMA-ES are that there are emitters of CMA-ES being trained in a cycle. Additionally, a feature map stores one elite for each possible feature configuration. Because there are invalid body designs, they do not produce an elite. When there is a successful robot design, the number of sensors, actuators, and body parts are used as features. If there are no elites or the current solution is better than the actual elite stored in the feature map, then the current solution is assigned to its feature configuration.

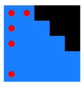
We use a slightly modified version of the CMA-ME with improvement emitters (Fontaine et al., 2020). We only restart an emitter when the number of elites is greater than the number of emitters and it is stuck for more than 500 generations. Being stuck means that the emitter could not find a better elite or an elite could not be placed into an empty feature configuration in the map. When an emitter restarts, the mean used to initialize the CMA-ES is a random elite in the map.

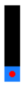
CMA-ME is executed for 60,000 generations for all environments, except the light chasing environment with 67,446 generations because we forced it to stop a longer training and its best fitness score was already better than the one trained with CMA-ES. The initial mean and the initial coordinate-wise standard deviation are the same as CMA-ES for all emitters. The population size is 128 because the CMA-ME training was executed in a computer with 128 threads at 2.9GHz.

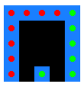
6 Results

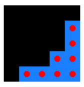
The training process took around 2.5 days for optimizing the NCA with CMA-ES. The evolution with CMA-ME took around 5.5 days. It is important to note that they do not have the same machine configuration, population size, and maximum number of generations.

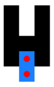
Fig. 4 shows all robot designs with the best fitness scores in regards to their training method and task. Almost all robots for the LC and CBT tasks fill the entire 5×5 grid of cells. Those environments do not have any environmental constraints (any obstacle) for the robot size. Therefore, we infer that the full grid of modules is easier to design and there are more computational resources for controlling the robot. Their fitness scores are shown in Table 1. The results indicate that CMA-ES and CMA-ME can reach almost the same fitness scores after training. However, CMA-ME has fewer generations for the 15 emitters (4,000 generations per emitter). It is possible that if we run 20,000 generations per emitter, CMA-ME could reach a better final performance than CMA-ES and with more diversity. The history of the maximum fitness score per generation is depicted in Fig. 5.


(a) CMA-ES - LC


(b) CMA-ES - LCO


(c) CMA-ES - CBT


(d) CMA-ME - LC


(e) CMA-ME - LCO

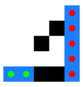

(f) CMA-ME - CBT

Figure 4: Robot designs with best fitness scores for the tasks of light chasing (LC), light chasing with obstacle (LCO) and carrying ball to target (CBT).

The elites were saved for both CMA-ES and CMA-ME, then we can compare their quality diversity. In Table 2, the number of cells filled and QD-scores of all six methods and tasks combinations are presented. It is noticeable that CMA-ME provides much more quality diversity because of its bigger number of feature configurations and its QD-score. We can visualize it in Fig. 6. This shows a small part of the elites produced for the light chasing tasks with CMA-ES and CMA-ME. Nevertheless, it confirms those two quality diversity measurements because more cells are filled, and there are more cells with higher fitness scores.

For testing the success of our six trained models, we run 100 times the simulation and the percentage of success is presented in Table 3. We can visualize some examples of those simulations in Fig. 7. The trained model with CMA-ES for the light chasing task got 92% of success rate with 0.58274 fitness score while the one trained with CMA-ME had 75% of success and 0.61481 of fitness score. This means that a higher fitness score does not indicate a more successful model for reaching the light. This can be observed in Fig. 7a-d for CMA-ES, and Fig. 7e-h for CMA-ME. We can see in Fig. 7g that the light is at the top-right corner and the robot goes to the top-left corner. This explains the 75% success rate of this NCRS because the light is at the top-right corner in 25 out of the 100 simulations. This model learned to move faster to the light in the other three corners, but it misses the one in the top-right corner. For the light chasing with obstacle task, the reason for the higher success rate of CMA-ES robot is that it is much thinner than the CMA-ME robot. Therefore, it is easier to pass through the passage. If we define success in LCO by passing the center of the body through the passage, then CMA-ES and CMA-ME had a success rate, respectively, of 77% and 45%. The NCRS did not learn to move to the light after passing through the obstacle. It just moves forward. Because of the difficulty of this task, we can consider the results for LCO were partially successful in general and successful in body design. Fig. 7i-l and Fig. 7m-p show that. The task of carrying a ball to a target had no successful trained model. The robots for both training methods just move forward and, by chance, it moves the ball to target. This can be seen in Fig. 7q-t and Fig. 7u-x.

Fig. 3 shows how the channels progress through time. The hidden channels are predominantly different in their behavior for the developmental and control phases. We infer this is mainly due to the control flag channel which regulates these two phases. We can observe the different patterns that emerged in their final time-steps. From the initial "seed" state to the state in Fig. 3a, we can see how the NCA behaves during 10 time-steps of the developmental phase. In Fig 3b, we can see the end of the control phase during its 200 time-steps (100 time-steps in the environment). We can still understand its behavior because the hidden and input/output channels were set to zero at the beginning of the control phase, and the body, control flag, tissue, sensor, and actuator channels were fixed according to the morphology of the robot.

7 Discussion and conclusion

Body-brain co-evolution is a challenging task (Bhatia et al., 2021). In this work, we developed three benchmark tasks for robot co-design and introduced a novel method by having a unified substrate as a genome with its own rules. This substrate is a single neural cellular automaton that works to develop and control a modular robot. This novelty opens up several possibilities in open-ended evolution (Stanley, 2019), especially because body and brain can co-evolve to the limits of the capacity of the artificial neural network. Because it defines the local rules in the CA, NCRS has the advantage of scalability. We also infer that curriculum learning will be important for complexifying the evolving robot (Bengio et al., 2009). For example, the number of body parts and dimensions can increase over time with the progress of the generations. Evolution in multi-agent environments may also be applied, such as in PolyWorld (Yaeger et al., 1994). We can also try to remove the two separated phases into one. Thus, we can observe how development and control can emerge and the performance the modular robots can have.

The presented results were successful for the LC task, but our trained models presented some failures when increasing the difficulty of the tasks. This may be addressed by adjusting the fitness score to reflect the success conditions, as well as by applying curriculum learning (Bengio et al., 2009). In future works, we plan to apply our method in the Evolution Gym (Bhatia et al., 2021), or in a modified version of VoxCraft (Liu et al., 2020) for 3D soft robots. Moreover, we aim at training and testing our approach for self-repair and robustness to noise.

Acknowledgment

This work was partially funded by the Norwegian Research Council (NFR) through their IKTPLUSS research and innovation action under the project Socrates (grant agreement 270961). We thank Henrique Galvan Debarba for his thoughtful comments about the text. We also thank Joachim Winther Pedersen, Djordje Grbic, Miguel González Duque, and Rasmus Berg Palm for the helpful discussions during the implementation of the experiments.

References

Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *Proceedings of the 26th annual international conference on machine learning*, pp. 41–48, 2009.

Jagdeep Bhatia, Holly Jackson, Yunsheng Tian, Jie Xu, and Wojciech Matusik. Evolution gym: A large-scale benchmark for evolving soft robots. *Advances in Neural Information Processing Systems*, 34, 2021.

Josh C Bongard and Rolf Pfeifer. Evolving complete agents using artificial ontogeny. In *Morpho-functional Machines: The new species*, pp. 237–258. Springer, 2003.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.

Luzius Brodbeck, Simon Hauser, and Fumiya Iida. Morphological evolution of physical robots through model-free phenotype development. *PloS one*, 10(6):e0128444, 2015.

John Conway et al. The game of life. *Scientific American*, 223(4):4, 1970.

René Doursat, Hiroki Sayama, and Olivier Michel. A review of morphogenetic engineering. *Natural Computing*, 12(4):517–535, 2013.

Sam Earle, Justin Snider, Matthew C Fontaine, Stefanos Nikolaidis, and Julian Togelius. Illuminating diverse neural cellular automata for level generation. *arXiv preprint arXiv:2109.05489*, 2021.

Matthew C Fontaine, Julian Togelius, Stefanos Nikolaidis, and Amy K Hoover. Covariance matrix adaptation for the rapid illumination of behavior space. In *Proceedings of the 2020 genetic and evolutionary computation conference*, pp. 94–102, 2020.

Agrim Gupta, Silvio Savarese, Surya Ganguli, and Li Fei-Fei. Embodied intelligence via learning and evolution. *Nature communications*, 12(1):1–12, 2021.

Nikolaus Hansen and Andreas Ostermeier. Adapting arbitrary normal mutation distributions in evolution strategies: The covariance matrix adaptation. In *Proceedings of IEEE international conference on evolutionary computation*, pp. 312–317. IEEE, 1996.

Nikolaus Hansen, Youhei Akimoto, and Petr Baudis. Cma-es/pycma on github. *Zenodo, doi*, 10, 2019.

Simon Harding and Wolfgang Banzhaf. Artificial development. In *Organic Computing*, pp. 201–219. Springer, 2009.

Donald J Hejna III, Pieter Abbeel, and Lerrel Pinto. Task-agnostic morphology evolution. *arXiv preprint arXiv:2102.13100*, 2021.

Kazuya Horibe, Kathryn Walker, and Sebastian Risi. Regenerating soft robots through neural cellular automata. In *EuroGP*, pp. 36–50, 2021.

Maciej Komosiński and Szymon Ulatowski. Framsticks: Towards a simulation of a nature-like world, creatures and evolution. In *European Conference on Artificial Life*, pp. 261–265. Springer, 1999.

Sam Kriegman, Nick Cheney, and Josh Bongard. How morphological development can guide evolution. *Scientific reports*, 8(1):1–10, 2018.

Christopher Langton. *Artificial life: proceedings of an interdisciplinary workshop on the synthesis and simulation of living systems*. Routledge, 2019.

Xia Li and Anthony Gar-On Yeh. Neural-network-based cellular automata for simulating multiple land use changes using gis. *International Journal of Geographical Information Science*, 16(4): 323–343, 2002.

Hod Lipson and Jordan B Pollack. Automatic design and manufacture of robotic lifeforms. *Nature*, 406(6799):974–978, 2000.

S Liu, D Matthews, S Kriegman, and J Bongard. Voxcraft-sim, a gpuaccelerated voxel-based physics engine, 2020.

Eric Medvet, Alberto Bartoli, Federico Pigozzi, and Marco Rochelli. Biodiversity in evolved voxel-based soft robots. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 129–137, 2021.

Alexander Mordvintsev, Ettore Randazzo, Eyvind Niklasson, and Michael Levin. Growing neural cellular automata. *Distill*, 5(2):e23, 2020.

Jean-Baptiste Mouret and Jeff Clune. Illuminating search spaces by mapping elites. *arXiv preprint arXiv:1504.04909*, 2015.

Stefano Nichele, Mathias Berild Ose, Sebastian Risi, and Gunnar Tufte. Ca-neat: evolved compositional pattern producing networks for cellular automata morphogenesis and replication. *IEEE Transactions on Cognitive and Developmental Systems*, 10(3):687–700, 2017.

Karl J Niklas and Stuart A Newman. The origins of multicellular organisms. *Evolution & development*, 15(1):41–52, 2013.

Justin K Pugh, Lisa B Soros, and Kenneth O Stanley. Quality diversity: A new frontier for evolutionary computation. *Frontiers in Robotics and AI*, 3:40, 2016.

Ettore Randazzo, Alexander Mordvintsev, Eyvind Niklasson, Michael Levin, and Sam Greydanus. Self-classifying mnist digits. *Distill*, 5(8):e00027–002, 2020.

Karl Sims. Evolving 3d morphology and behavior by competition. *Artificial life*, 1(4):353–372, 1994.

Jonathan MW Slack and Leslie Dale. *Essential developmental biology*. John Wiley & Sons, 2021.

Kenneth O Stanley. Compositional pattern producing networks: A novel abstraction of development. *Genetic programming and evolvable machines*, 8(2):131–162, 2007.

Kenneth O Stanley. Why open-endedness matters. *Artificial life*, 25(3):232–235, 2019.

Shyam Sudhakaran, Djordje Grbic, Siyan Li, Adam Katona, Elias Najarro, Claire Glanois, and Sebastian Risi. Growing 3d artefacts and functional machines with neural cellular automata. *arXiv preprint arXiv:2103.08737*, 2021.

Jacopo Talamini, Eric Medvet, and Stefano Nichele. Criticality-driven evolution of adaptable morphologies of voxel-based soft-robots. *Frontiers in Robotics and AI*, 8:172, 2021.

Pawel Topa. Network systems modelled by complex cellular automata paradigm. *Cellular automata-simplicity behind complexity. InTech, Europe, China*, pp. 259–274, 2011.

Alexandre Variengien, Sidney Pontes-Filho, Tom Eivind Glover, and Stefano Nichele. Towards self-organized control: Using neural cellular automata to robustly control a cart-pole agent. *Innovations in Machine Intelligence*, 1:1–14, Dec 2021. doi: 10.54854/imi2021.01.

Frank Veenstra and Kyrre Glette. How different encodings affect performance and diversification when evolving the morphology and control of 2d virtual creatures. In *ALIFE: proceedings of the artificial life conference*, pp. 592–601. MIT Press, 2020.

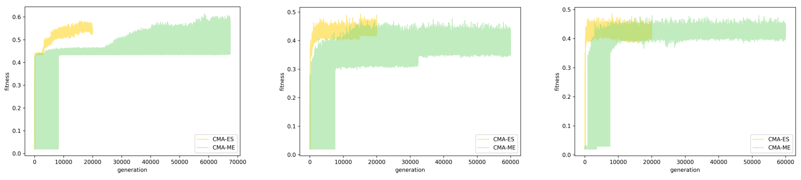
Stephen Wolfram. *A new kind of science*, volume 5. Wolfram media Champaign, IL, 2002.

Larry Yaeger et al. 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-PROCEEDINGS VOLUME-*, volume 17, pp. 263–263. Citeseer, 1994.

Mark Yim, Wei-Min Shen, Behnam Salemi, Daniela Rus, Mark Moll, Hod Lipson, Eric Klavins, and Gregory S Chirikjian. Modular self-reconfigurable robot systems [grand challenges of robotics]. *IEEE Robotics & Automation Magazine*, 14(1):43–52, 2007.

Anthony M Zador. A critique of pure learning and what artificial neural networks can learn from animal brains. *Nature communications*, 10(1):1–7, 2019.

A Appendix



(a) Light chasing (b) Light chasing with obstacle (c) Carrying ball to target

Figure 5: Maximum fitness score through generations.

Table 2: Elites stored during training for the light chasing (LC), light chasing with obstacle (LCO) and carrying ball to target (CBT)

	CMA-ES		CMA-ME	
	Cells filled	QD-score	Cells filled	QD-score
LC	67.58%	0.29530	89.57%	0.40152
LCO	17.88%	0.06069	61.80%	0.19841
CBT	58.10%	0.22957	93.82%	0.37996

Table 3: Testing success percentage over 100 runs for the tasks of light chasing (LC), light chasing with obstacle (LCO) and carrying ball to target (CBT)

	CMA-ES	CMA-ME
LC	92%	75%
LCO	20%	8%
CBT	1%	2%

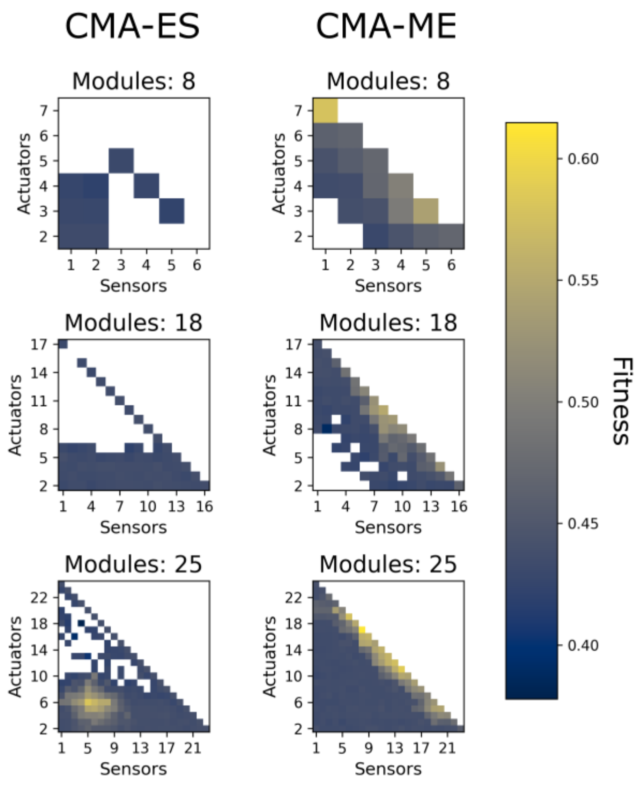
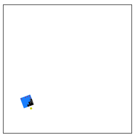
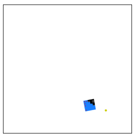
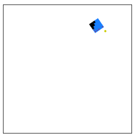
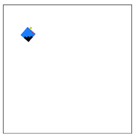


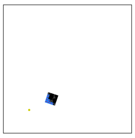
Figure 6: Selected elites trained in the light chasing environment. Those modules were selected because they are the most different between CMA-ES and CMA-ME. Axes and subplots indicate the number of components.

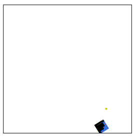

(a) CMA-ES - LC #1

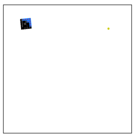

(b) CMA-ES - LC #2

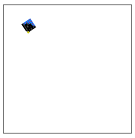

(c) CMA-ES - LC #3

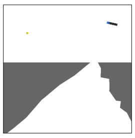

(d) CMA-ES - LC #4

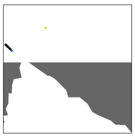

(e) CMA-ME - LC #1

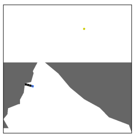

(f) CMA-ME - LC #2

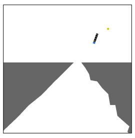

(g) CMA-ME - LC #3

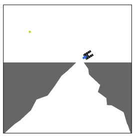

(h) CMA-ME - LC #4

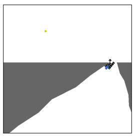

(i) CMA-ES - LCO #1

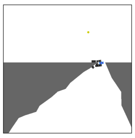

(j) CMA-ES - LCO #2

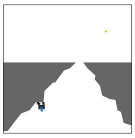

(k) CMA-ES - LCO #3

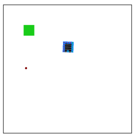

(l) CMA-ES - LCO #4

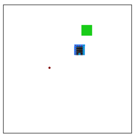

(m) CMA-ME - LCO #1

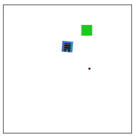

(n) CMA-ME - LCO #2

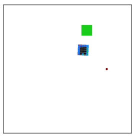

(o) CMA-ME - LCO #3

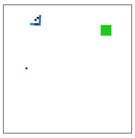

(p) CMA-ME - LCO #4

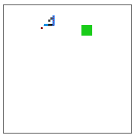

(q) CMA-ES - CBT #1

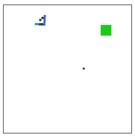

(r) CMA-ES - CBT #2


(s) CMA-ES - CBT #3


(t) CMA-ES - CBT #4


(u) CMA-ME - CBT #1


(v) CMA-ME - CBT #2


(w) CMA-ME - CBT #3

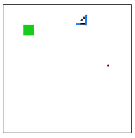

(x) CMA-ME - CBT #4

Figure 7: Last time-step where the robot is fully visible of the best NCRS trained with CMA-ES and CMA-ME in the environments for light chasing (LC), light chasing with obstacle (LCO) and carrying ball to target (CBT).

Paper C.3

Collective control of modular soft robots via embodied spiking neural cellular automata
(Nadizar et al., 2022)

Author(s):
Giorgia Nadizar, Eric Medvet, Stefano Nichele and Sidney Pontes-Filho

Published at workshop:
From Cells to Societies: Collective Learning Across Scales at Tenth International Conference on Learning Representations (ICLR 2022)

Paper C.3
Nadizar et al. (2022)

Collective control of modular soft robots via embodied Spiking Neural Cellular Automata

Giorgia Nadizar & Eric Medvet
University of Trieste, Trieste, Italy
giorgia.nadizar@phd.units.it,emedvet@units.it

Stefano Nichele & Sidney Pontes-Filho
Oslo Metropolitan University, Oslo, Norway
{stenic,sidneyp}@oslomet.no

Abstract

Voxel-based Soft Robots (VSRs) are a form of modular soft robots, composed of several deformable cubes, i.e., voxels. Each VSR is thus an ensemble of simple agents, namely the voxels, which must cooperate to give rise to the overall VSR behavior. Within this paradigm, collective intelligence plays a key role in enabling the emerge of coordination, as each voxel is independently controlled, exploiting only the local sensory information together with some knowledge passed from its direct neighbors (distributed or collective control). In this work, we propose a novel form of collective control, influenced by Neural Cellular Automata (NCA) and based on the bio-inspired Spiking Neural Networks: the embodied Spiking NCA (SNCA). We experiment with different variants of SNCA, and find them to be competitive with the state-of-the-art distributed controllers for the task of locomotion. In addition, our findings show significant improvement with respect to the baseline in terms of adaptability to unforeseen environmental changes, which could be a determining factor for physical practicability of VSRs.

1 Introduction and related works

Biological organisms are intrinsically modular at different scales (Lorenz et al., 2011). The collective self-organization at the cellular level results in the emergence of complex bodies and brains without any form of centralized control. Such modularity allows for mechanisms of local interaction which, in turn, result in collective learning and adaptation.

In the artificial domain, modular robotics (Alattas et al., 2019) provides a framework for the investigations of biologically-inspired principles of collective control through distributed coordination of the agents composing the robot (Cheney et al., 2014). In addition, modular robots allow for a high degree of reconfiguration and self-assembly (Pathak et al., 2019), as well as fault tolerance and modules reusability. However, in order to exploit such opportunities, there is the need for modular distributed controllers, possibly embedded in each module. Therefore, the overall behavior of the robot is the result of the collective interplay of distributed sensing, local communication, and actuation of interacting body modules. In addition, identical modules would facilitate the reusability of the parts and robustness in case of damage (Huang et al., 2020). In this work, we focus on a specific type of modular robots, namely Voxel-based Soft Robots (VSRs) (Hiller & Lipson, 2012). Since VSRs are robots made of interconnected soft blocks (voxels), each module may be considered an agent in a collective. As such, mechanisms of collective intelligence are desired. While such mechanisms of collective intelligence are rather popular in the context of swarm robotics (Hamann, 2018), e.g., via self-assembly of thousands of robots through local interactions (Rubenstein et al., 2014), they are less explored in the context of modular robotics.

One paradigm of distributed neural control through local interactions of identical cells is Neural Cellular Automata (NCA) (Li & Yeh, 2002; Nichele et al., 2017; Mordvintsev et al., 2020). In case of robots composed of identical modules, each NCA cell can be embodied in a robot module. Such approach would in theory facilitate a physical realization as no global wiring nor centralized control would be needed. NCA have been successfully used to grow and replicate CA shapes and structures with neuroevolution (Nichele et al., 2017) and with differentiable learning (Mordvintsev et al., 2020), to produce self-organising textures (Niklasson et al., 2021), to grow 3D artifacts (Sudhakaran et al., 2021), for regenerating soft robots (Horibe et al., 2021), and for controlling reinforcement learning agents (Variengien et al., 2021).

In this work, we introduce a novel embodied NCA model based on more biologically plausible Spiking Neural Network (SNN) controllers. We name it embodied Spiking Neural Cellular Automata (SNCA). SNNs incorporate neuronal and synaptic states in their neuron models, as well as the concept of time. SNCA open up several opportunities in the domain of modular robotics, such as mechanisms of homeostatic adaptation and local learning rules, e.g., spike-timing-dependent plasticity. In addition, nearby modules communicate natively through spikes which are not generated at every clock-cycle but only when the internal neural membrane potential reaches a specific threshold, which in turn changes the membrane potential of the post-synaptic neuron, either within the same robotic module or in a nearby module in case of modular robots. The advent of neuromorphic hardware, which natively supports SNNs execution and learning, may provide orders of magnitude improvement in energy consumption as compared to traditional neural networks (Blouw et al., 2019). Low energy consumption is considered to be an enabling factor for the physical realization of self-organizing VSRs.

This work is organized as follows: in Section 2 we introduce VSRs, their morphology, and the proposed NCA-based controllers. In Section 3 we describe SNNs and SNCA. In Section 4 we present the experimental results and discuss the insights they provide. Finally, in Section 5 we draw the conclusions.

2 Collective control of Voxel-Based Soft Robots

Voxel-Based Soft Robots (Hiller & Lipson, 2012) are a kind of modular soft robots, composed of several elastically deformable cubes (*voxels*). In this work, we experiment with a 2D version of VSRs, simulated in discrete time and continuous space (Medvet et al., 2020b). The way in which VSRs achieve movement is a direct consequence of the unique combination of softness and modularity. The global behavior derives from the collective and synergistic contraction and expansion of individual voxels, similarly to what happens in biological muscles. Because of modularity, a VSR can be considered as an ensemble of simple sub-agents, the voxels, which are physically joined to obtain a greater structure, and whose individual behaviors influence eachother and concur to the emergence of coordination. Therefore, to characterize a VSR we need to specify how the voxels are assembled and what are their properties (*morphology*), and how the voxels compute their control signals and communicate with one another (*controller*).

2.1 VSR morphology: assembling individual voxels

The morphology of a VSR specifies how individual voxels are assembled, their sensory equipment, and their physical properties. A VSR can be represented as a 2D grid of voxels, describing their spatial organization and assembly. Adjacent voxels in the grid are rigidly linked: not only does this allow to assemble a robot out of primitive modules, but it also forces mutual physical interactions resulting in an overall complex dynamics. In addition, each voxel can be equipped with sensors to enable proprioception and awareness of the surroundings. For each voxel, we use three types of sensors: (a) *area* sensors, perceiving the ratio between the current area of the voxel and its rest area, (b) *touch* sensors, sensing if the voxel is in contact with the ground or with another body, and (c) *velocity* sensors, which perceive the velocity of the center of mass of the voxel along the x- and y-axes. We normalize sensor readings to be defined in $[0, 1]^4$.

Concerning physical properties, we model voxels as compounds of spring-damper systems, masses, and distance constrains (Medvet et al., 2020c), whose parameters can be changed to alter features like elasticity or actuation power. Each voxel changes volume (actually area, in the 2D case) over time, due to passive interactions with other bodies and the active application of a control signal. The latter is computed at each simulation time step and is defined in $[-1, 1]$, where -1 corresponds to maximum requested expansion and 1 corresponds to maximum requested contraction. In the employed model (Medvet et al., 2020b), contraction and expansion correspond to linear variations of the rest-length of the spring-damper system, proportional to the control signal received.

2.2 VSR CONTROLLER: THE EMBODIED NEURAL CELLULAR AUTOMATA PARADIGM

A VSR controller derives from the ensemble of individual voxels controllers. However, achieving coordination while keeping the intelligent control of each voxel fully independent of the others is a difficult task. In fact, most studies involving VSRs either rely on independent, yet not intelligent, controllers based on trivial sinusoidal functions (Hiller & Lipson, 2012; Corucci et al., 2018; Kriegman et al., 2018), or sacrifice modularity and deploy a central neural controller that has access to all voxels (Talamini et al., 2019; Ferigo et al., 2021; Nadizar et al., 2021; 2022). To solve this issue, Medvet et al. (2020a) introduced the concept of distributed neural controllers, which exploit message passing between neighbors to allow the emerge of coordination thanks to collective intelligence. Here, we follow along the same direction, combining the key ideas of modularity and collective intelligence, with an approach based on Neural Cellular Automata (NCA) techniques (Li & Yeh, 2002; Nichele et al., 2017; Mordvintsev et al., 2020), in which the lookup table of each Cellular Automaton (CA) cell is replaced by an Artificial Neural Network (ANN). More in detail, we consider each voxel as a single *embodied* NCA cell (from now on, simply referred to as NCA), which has access to the local sensor readings and to some information coming from the neighbors, to compute the local actuation value and the messages directed towards adjacent voxels. Our approach, anyhow, has a substantial difference with the standard NCA architectures: namely, it is strongly bound to the VSR morphology employed, as we only instantiate NCA cells in correspondence to the voxels.

Formally, at every time step k, each NCA takes as input a vector $\boldsymbol{x}^{(k)} = \left[\boldsymbol{r}^{(k)}\ \boldsymbol{i}_{\uparrow}^{(k)}\ \boldsymbol{i}_{\leftarrow}^{(k)}\ \boldsymbol{i}_{\downarrow}^{(k)}\ \boldsymbol{i}_{\rightarrow}^{(k)}\right]$ and produces as output a vector $\boldsymbol{y}^{(k)} = \mathrm{ANN}_{\boldsymbol{\theta}}\left(\boldsymbol{x}^{(k)}\right) = \left[a^{(k)}\ \boldsymbol{o}_{\uparrow}^{(k)}\ \boldsymbol{o}_{\leftarrow}^{(k)}\ \boldsymbol{o}_{\downarrow}^{(k)}\ \boldsymbol{o}_{\rightarrow}^{(k)}\right]$ where $\boldsymbol{r}^{(k)} \in \mathbb{R}^4$ is the local sensor reading, $\boldsymbol{i}_{\uparrow}^{(k)}, \boldsymbol{i}_{\leftarrow}^{(k)}, \boldsymbol{i}_{\downarrow}^{(k)}, \boldsymbol{i}_{\rightarrow}^{(k)}$, each defined in $\mathbb{R}^{n_c}$, are values coming from adjacent voxels (from above, left, below, right, respectively, and set to $\mathbf{0} \in \mathbb{R}^{n_c}$ if no voxel is present in the corresponding direction), $a^{(k)}$ is the actuation value, $\boldsymbol{o}_{\uparrow}^{(k)}, \boldsymbol{o}_{\leftarrow}^{(k)}, \boldsymbol{o}_{\downarrow}^{(k)}, \boldsymbol{o}_{\rightarrow}^{(k)}$, each defined in $\mathbb{R}^{n_c}$, are values directed to adjacent voxels (to above, left, below, right, respectively), and $\boldsymbol{\theta}$ are the parameters of the ANN constituting the NCA. Values output by a NCA at time k are used by an adjacent NCA at time $k+1$, e.g., given a NCA a that outputs $\boldsymbol{o}_{a,\rightarrow}^{(k)}$, the NCA b at its right will have $\boldsymbol{i}_{b,\leftarrow}^{(k+1)} = \boldsymbol{o}_{a,\rightarrow}^{(k)}$.

We experiment with three ways of instantiating the general scheme described above, *non-uniform directional* ($\cancel{\mathrm{U}}$D), *uniform directional* (UD), and *uniform non-directional* (U$\cancel{\mathrm{D}}$), which differ in the homogeneity of the individual cells (uniform vs. non-uniform) and in the information passed between voxels (directional vs. non-directional). The first two schemes are inspired by already existing forms of distributed controllers (Medvet et al., 2020a) and we consider them as baselines, whereas the U$\cancel{\mathrm{D}}$-NCA is novel.

The most evident, yet conceptually simple, difference lies in the *uniformity*: in $\cancel{\mathrm{U}}$-NCA, cells have a different ANN in each voxel (each with parameters $\boldsymbol{\theta}_i$), whereas in U-NCA all cells ANNs share the same parameters $\boldsymbol{\theta}$. Therefore, it follows that, for a given ANN architecture, the amount of parameters of $\cancel{\mathrm{U}}$-NCA is n_{voxels} times the amount of parameters of U-NCA.

The second distinguishing element is *directionality*. In $\cancel{\mathrm{D}}$-NCA, cells send the same output to all the adjacent cells, i.e., $\boldsymbol{o}_{\uparrow}^{(k)} = \boldsymbol{o}_{\leftarrow}^{(k)} = \boldsymbol{o}_{\downarrow}^{(k)} = \boldsymbol{o}_{\rightarrow}^{(k)} = \boldsymbol{o}^{(k)}$ and $\boldsymbol{y}^{(k)} = \left[a^{(k)}\ \boldsymbol{o}^{(k)}\right]$, whereas in D-NCA, cells send, in general, different outputs. The D-NCA hence corresponds to the one originally proposed by Medvet et al. (2020a). Contrarily, $\cancel{\mathrm{D}}$-NCA are more adherent to the original concept of NCA as $\boldsymbol{o}^{(k)}$ can be interpreted as the current state of the cell.

The proposed types of NCA controllers can be employed with any type of ANN, either fully-connected feed-forward NNs, i.e., multi-layer perceptrons (MLP), or more biologically plausible NNs, such as the SNNs described in the following section.

3 SPIKING NEURAL NETWORKS AS ROBOTIC CONTROLLERS

Spiking Neural Networks (SNNs) are a type of ANNs in which biological resemblance plays a fundamental role (Gerstner & Kistler, 2002). Often referred to as the third generation of ANNs (Maass, 1997), SNNs are characterized by a more biologically and neuroscientifically faithful neural model than classical ANNs.

The key element of SNNs is the modeling of the evolution over time of the membrane potential of neurons. Modifications of such potential are caused by incoming neural stimuli, which can either be excitatory (increasing the potential) or inhibitory (decreasing it). Neural stimuli occur in the form of spikes over time, which can propagate along synapses in order to reach different neurons of the SNN, enabling information passing within the network. The generation of said spikes is called *firing*, and arises whenever the membrane potential of a neuron exceeds a given threshold. Despite the binary nature of spikes, the intensity of any stimulus received by a neuron is modulated by the strength of the synapse connecting the firing neuron (pre-synaptic neuron) and the neuron receiving the spike (post-synaptic neuron). Not unlike classical MLPs, synapses are modeled as weighted connections, where the weights play the main role in determining the behavior of the ANN, and can be subject to task-oriented optimization.

What is indeed essentially different between MLPs and SNNs, is the way information is encoded, which is a direct consequence of the peculiarities of the two models. In particular, in MLPs there is no notion of time, and information is encoded in the form of real values traveling along the synapses. Conversely, SNNs are bound to the concept of time to compute the evolution of the neural membranes and for the propagation of spikes in the network. Within this framework, information is embedded in the time distribution of spikes. Hence, additional tools are required to interpret spike trains as real values and vice versa.

Given their high biological resemblance, SNNs are extremely promising robotic controllers. In fact, faithfully mimicking the functioning of the nervous systems of living organisms could be an enabling factor for bringing the desirable traits of biological organisms to artificial agents, e.g., autonomy or adaptability. Moreover, the possibility of deploying SNNs on highly energy efficient neuromorphic hardware (Li et al., 2014) is an additional profitable feature, which could be of paramount importance with reference to energy constraints.

3.1 DISCRETE TIME LEAKY INTEGRATE AND FIRE MODEL

Several spiking neuron models have been proposed (Izhikevich, 2004), which, despite differing in terms of biological plausibility and computational costs, all share the main concepts derived from neuroscience. Among them, we employ the computationally efficient Leaky Integrate and Fire (LIF) model, simulated in discrete time. The LIF model represents the neural membrane as a capacitor, whose potential can be increased or decreased by inputs (excitatory or inhibitory), and exponentially decays with time. At each neural simulation time step h, the membrane potential $v_j^{(h)}$ of a LIF neuron j is updated as:

$$v_j^{(h)} = v_j^{(h-1)} + \sum_{i=1}^{n} w_{i,j} s_i^{(h)} - \Delta t_h \lambda_v v^{(h-1)}, \tag{1}$$

with $w_{i,j} \in \mathbb{R}$ being the synaptic weight of the i-to-j synapse, n being the number of incoming synapses, $s_i^{(h)} \in \{0, 1\}$ carrying pre-synaptic neuron spike, and $\Delta t_h = 1/f_h$ being the neural simulation time resolution. After the update, and if the membrane potential $v_j^{(h)}$ exceeds a threshold $\vartheta_j^{(h)}$, the neuron j outputs a spike, i.e., $s_j^{(h)} = 1$, and the membrane potential is reset to its resting value v_{rest}, otherwise $s_j^{(h)} = 0$.

We enhance the LIF model introducing the biological concept of homeostatic plasticity. Homeostasis is a self regulatory mechanism present at various sites of living organisms, which aims at re-establishing equilibrium in contrast to strong stimuli that could unbalance a system (Turrigiano & Nelson, 2004). In our case, homeostasis operates as a firing rate regulator, acting on the threshold $\vartheta_j^{(h)}$ of neurons, to prevent excessive or too scarce activity:

$$\vartheta_j^{(h)} = \min\left(\vartheta_j^{(h-1)}, \sum_{i=1}^{n} w_{i,j}\right) + \psi_j^{(h)}, \tag{2}$$

with $\psi_j^{(h)}$ being a parameter updated as:

$$\psi_j^{(h)} = \begin{cases} \psi_j^{(h-1)} + \psi_{\text{inc}} & \text{if } s_j^{(h-1)} = 1 \\ \psi_j^{(h-1)} - \psi_j^{(h-1)} \lambda_\psi \Delta t_h & \text{otherwise.} \end{cases} \tag{3}$$

3.2 The LIF model inside NCA

We employ the LIF model described above within the ANN for the NCA in our robots. We simulate both the robot mechanical models and the LIF SNNs in discrete time: however, we update the simulation of the LIF SNNs with a greater frequency. Namely, we update the mechanical model with a frequency $f_k = 60\,\text{Hz}$ (the default value of the 2D-VSR-Sim by Medvet et al. (2020b)) and the SNNs with a frequency $f_h = 16 f_k \approx 1\,\text{kHz}$ (as suggested by Izhikevich (2004)).

In practice, at each k, we build a spike train $\left(s^{(16k)}, \dots, s^{(16k+15)}\right) \in \{0,1\}^{16}$ to be fed to the SNNs from each element of the sensor reading $\boldsymbol{r}^{(k)}$ and we compute the actuation value $a^{(k)}$ considering the spikes emitted by the corresponding SNN output neuron up to $h = 16k$. Concerning the information traveling between pairs of SNNs, we simply copy the spike trains with a delay of 16 time steps in the SNN simulation, i.e., one time step in the robot simulation, consistently with the description given in Section 2.2. For performing the sensor reading and actuation value conversions, we take inspiration from rate coding (Bouvier et al., 2019), where real values are mapped to a frequencies, which are then used to generate spike trains (Wang et al., 2008).

For spike trains to be fed to input neurons corresponding to sensor readings, we set:

$$s^{(h)} = \begin{cases} 1 & \text{if } \exists n \in \mathbb{N} \text{ s.t. } h = h_{\text{last}} + n \left\lfloor \frac{f_h}{f^{(k)}} \right\rfloor \\ 0 & \text{otherwise,} \end{cases} \tag{4}$$

where h_{last} is the time step of the last spike to the neuron (initially set to 0) and $f^{(k)} = r^{(k)}(f_{\text{max}} - f_{\text{min}}) + f_{\text{min}}$, $r^{(k)} \in [0,1]$ being the element of the sensor reading. That is, we first linearly scale the scalar input to a frequency $f^{(k)} \in [f_{\text{min}}, f_{\text{max}}]$ and then we emit spikes at frequency $f^{(k)}$, i.e., one spike each $\left\lfloor \frac{f_h}{f^{(k)}} \right\rfloor$ time steps of the SNN simulation. We set $f_{\text{min}} = 5\,\text{Hz}$ and $f_{\text{max}} = 50\,\text{Hz}$ for biological plausibility: hence, with the maximum scalar input $r^{(k)} = 1$, we emit one spike each $\frac{1\,\text{kHz}}{50\,\text{Hz}} = 20$ time steps, whereas with the minimum input $r^{(k)} = 0$, we emit one spike each $\frac{1\,\text{kHz}}{5\,\text{Hz}} = 200$ time steps.

For the actuation value, we set:

$$a^{(k)} = 2\left(\frac{1}{f_{\text{max}}} \frac{f_h}{n_w} \sum_{k'=k-n_w+1}^{k} \sum_{h=16k'}^{16k'+15} s^{(h)}\right) - 1. \tag{5}$$

That is, we count the spikes in the last n_w robot simulation time steps, we linearly scale this value to $[0,1]$ considering the maximum possible frequency f_{max}, and then we linearly scale to $[-1,1]$. The reason why we consider n_w robot simulation time steps, instead of just the current one, is to have a better resolution of the actuation value and, hence, a smoother control. After preliminary experimentation, we set $n_w = 5$.

4 Experiments and results

We performed an extensive experimental campaign to investigate how coordination can emerge from different forms of collective control. We aimed at evaluating if we could improve the baselines of distributed control for VSRs (Medvet et al., 2020a) with our novel contribution, the U$\cancel{D}$-SNCA.

Therefore, we addressed the following research question: *"are U$\not{D}$-SNCA superior with reference to the baselines of distributed control?"*. To determine the effectiveness of a collective controller, we deployed it onto a VSR, and optimized its parameters using as quality measure the velocity achieved by the robot performing locomotion on a flat terrain. In addition, we also assessed the controllers adaptability, by measuring the VSR velocity, after the optimization, on a set of unseen terrains, i.e., terrains not used to optimize the controller parameters. With the extent of obtaining more general results, we experimented with three different morphologies.

4.1 U$\not{D}$-SNCA VS. BASELINE EMBODIED NCA

In order to provide an answer to the posed research question, we started by optimizing the parameters of three variants of NCA for each of the three considered morphologies, for a total of nine VSRs optimizations.

Concerning the NCA, we took into consideration the UD- and $\not{U}$D-NCA as baselines, and we compared them against the U$\not{D}$-SNCA. We used a MLP with $\tanh$ as activation function for both baseline NCA, while we equipped the SNCA with a fully-connected feed-forward SNN based on the LIF neural model augmented with homeostasis, with the following parameters: $v_{\text{rest}} = 0\,\text{mV}$, $\lambda_v = 0.01\,\text{s}^{-1}$, $\vartheta^{(0)} = 1\,\text{mV}$, $\psi^{(0)} = 0\,\text{mV}$, $\psi_{\text{inc}} = 0.2\,\text{mV}$, $\lambda_\psi = 0.01\,\text{s}^{-1}$. For both ANNs, we used 1 hidden layer, setting its size equal to the size of the input layer. We set $n_c = 1$ for both UD- and $\not{U}$D-NCA, and $n_c = 4$ for the U$\not{D}$-SNCA, in order to make the sizes of the ANNs output layers equal. Our choice of NCA hyper-parameters was driven by some exploratory experiments and by previous work involving SNNs (Pontes-Filho & Nichele, 2019) and NCA (Nichele et al., 2017; Mordvintsev et al., 2020).

Regarding the morphologies, we experimented with 3 VSRs, a biped , a comb , and a worm . We chose these morphologies to test the NCA controllers versatility, because they resemble different forms of living organisms, which take advantage of their diverse body shapes to achieve diversified gaits. We relied on 2D-VSR-Sim (Medvet et al., 2020c) for the VSRs simulation, leaving all parameters to their default values. We made the code for the experiments publicly available at `https://github.com/giorgia-nadizar/VSRCollectiveControlViaSNCA`.

To optimize the NCA parameters, we resorted to evolutionary algorithms (EAs) as they can easily overcome the struggles posed by the non-differentiability in SNNs. In addition, EAs are well suited for ill-posed problems with many local optima, which makes them particularly appropriate for optimizing the parameters of robotic controllers. In this study, we used a simple form of evolutionary strategy (ES). At first, the population is initialized with n_{pop} individuals, i.e., numerical vectors $\boldsymbol{\theta}$, all generated by assigning to each element of the vector a randomly sampled value from a uniform distribution over the interval $[-1, 1]$. Subsequently, n_{gen} evolutionary iterations are performed, until reaching a total of n_{evals} fitness evaluations. On every iteration, the fittest quarter of the population is chosen to generate $n_{\text{pop}} - 1$ children, each obtained by adding values sampled from a normal distribution $N(0, \sigma)$ to each element of the element-wise mean $\boldsymbol{\mu}$ of all parents. The generated offspring, together with the fittest individual of the previous generation, end up forming the population of the next generation, which maintains the fixed size n_{pop}. We used the following ES parameters: $n_{\text{pop}} = 36$, $n_{\text{evals}} = 30\,000$, and $\sigma = 0.35$. We verified that evolution was in general capable of converging to a solution with the chosen values, despite the different sizes of the search spaces corresponding to each NCA configuration.

We optimized VSRs for the task of *locomotion* on a flat terrain, the goal being traveling as fast as possible along the positive x-axis. We assessed the performance of a VSR by measuring its average velocity v_x along the x-axis during a simulation of $30\,\text{s}$. We discarded the first $5\,\text{s}$ of each simulation to exclude the initial transitory from the velocity measurements. We used v_x as fitness measure for selecting the best individuals in the ES. For each of the 9 VSRs resulting from the combination of 3 NCA and 3 morphologies, we performed 10 independent evolutionary optimizations, i.e., with different random seeds, for a total of 90 runs.

Besides testing the VSR effectiveness upon parameters optimization, i.e., at the end of evolution, we also appraised their adaptability. We define a VSR controller as *adaptable*, if it is able to achieve good performance in locomotion in spite of environmental changes. To evaluate this in practice, we took each optimized VSR and re-assessed it on a set of unseen terrains, i.e., terrains which none of its ancestors ever experienced locomotion on. In particular, we experimented with the following

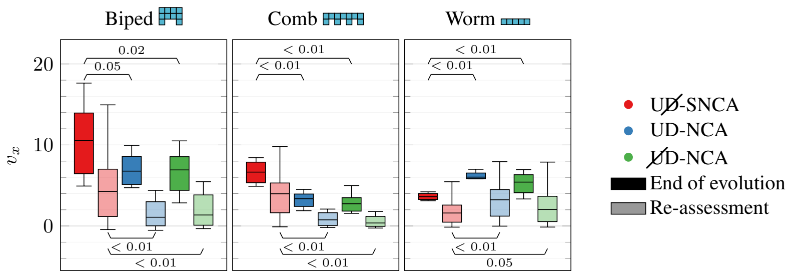


Figure 1: Box plots of the velocities v_x achieved by the best individuals at the end of evolution and upon re-assessment on unseen terrains for different VSR morphologies (plot columns) and embodied NCA controllers (color). Above pairs of boxes we report the p-values resulting from Mann-Whitney U tests with the null hypothesis of equality of the means.

terrains: (a) hilly with 6 combinations of heights and distances between hills, (b) steppy with 6 combinations of steps heights and widths, (c) downhill with 2 different inclinations, and (d) uphill with 2 different inclinations. As a result, we re-assessed each individual on a total of 16 different terrains; we define its adaptability as the average of the v_x on those terrains (each computed in a 30 s simulation, discarding the initial 5 s).

The results of our experimental evaluation are summarized in Figure 1. More in detail, for each of the considered VSR morphologies and NCA variants, we display the distribution of velocities achieved at the end of evolution by the best individuals, and their performance in terms of adaptability, i.e., the distribution of their average velocity on unseen terrains. In addition, above pairs of box plots, we report the p-values resulting from a two-sided Mann Whitney U statistical test; we consider, unless otherwise specified, $\alpha = 0.05$ as confidence level.

From Figure 1, we observe that for the biped and the comb morphologies, U~~D~~-SNCA are always significantly better than the baseline in terms of adaptability, and they are significantly better at the end of evolution in all but one case. However, the outcomes seem to be exactly opposite for the worm morphology, making it difficult to provide a general answer to the posed research question.

To further investigate on this apparent contradiction, we examined the behavior of a few evolved VSRs (videos are available at `https://giorgia-nadizar.github.io/VSRCollectiveControlViaSNCA/`) and found the reason behind the failure of the SNCA to be glaring for the worm morphology. In particular, we noticed that the NCA based on MLPs trigger a high frequency dynamics, resulting in a vibrating behavior, which for SNCA is prevented by homeostasis and $n_w = 5$ when converting spikes to actuation values. However, for the worm morphology, vibration appears to be the only effective gait, as these VSRs are not able to properly bend, having only one row of voxels at disposal. Conversely, the biped and comb morphologies have more complex structures, which allow the discovery of a wider range of efficacious gaits. In fact, when we inspected the behaviors of these two families of VSRs, we could notice a broader variety of gaits, with some tendencies to vibration among those controlled by MLP-based NCA. Avoiding vibrating behaviors, which have been shown to be a strong attractor in evolution (Medvet et al., 2021), is of paramount importance, as this type of movement severely hinders adaptability, and constitutes an insurmountable barrier for the physical practicability of VSRs, i.e., a form of reality gap (van Diggelen et al., 2021; Salvato et al., 2021). Even though it is possible to explicitly discourage vibrating behaviors, e.g., decreasing the actuation frequency, having a controller which avoids them by design is an undeniably significant accomplishment.

4.2 Strengths of the uniform non-directional SNCA

From the experimental outcomes, however, another question arises: *"what are the reasons behind the success of the U$\not D$-SNCA?"*. Namely, does the improvement lay in the non-directionality of the NCA or in the SNN employed?

To address the newly emerged points, we deepened our analysis with a supplementary experimental campaign, encompassing new combinations of NCA architectures, neural models, and morphologies, for a total of 12 additional VSRs to be optimized. Regarding the morphologies, we experimented with the biped and the comb, discarding the worm for the reasons highlighted in Section 4.1. For what concerns the controllers, we extended the previous experiments by evaluating all missing combinations of NCA architecture and neural models. Among the latters, we also included a SNN composed of LIF neurons for which we disabled homeostasis, keeping the value of the threshold fixed throughout the simulation to its initial value $\vartheta_i^{(h)} = \vartheta_i^{(0)} = 1\,\mathrm{mV}$. For each of the 12 new VSRs we repeated the experimental pipeline of Section 4.1.

We display the results, together with the outcomes of the previous experiments, in Table 1. Each cell of the table reports the median of the velocities achieved by VSRs at the end of evolution and upon re-assessment, grouped by morphology; each row corresponds to a NCA architecture, whereas we put neural models on the columns. We color cells proportionally to the median of velocities in order to better convey the information.

	End of evolution						Re-assessment					
	Biped ▦			Comb ▦			Biped ▦			Comb ▦		
	S-H	S	M	S-H	S	M	S-H	S	M	S-H	S	M
UD	6.5	5.5	8.0	3.8	5.9	3.8	3.7	2.3	2.2	2.6	3.8	1.4
$\not U$D	10.2	7.0	7.9	5.8	2.8	3.4	5.5	2.9	3.4	3.5	1.4	1.2
U$\not D$	13.1	13.0	9.3	7.5	5.9	8.5	5.5	5.5	2.2	5.1	3.9	4.7

Table 1: Medians of velocities v_x achieved by the best individuals at the end of evolution and upon re-assessment on unseen terrains for different morphologies. We put different NCA architectures on each row, and ANN models on the columns (M stands for MLP, S for SNN without homeostasis, S-H for SNN with homeostasis). Cells are colored proportionally to v_x.

From examining Table 1, we can investigate on the importance of the two aforementioned factors. First, to weigh the impact of the NCA architecture, we compare the medians of different rows for each column of the table. We observe that U$\not D$-NCA are not worse than both D-NCA in all but one case, and they always equal or outperform D-NCA if combined with SNNs. We speculate this descends from the fact that, especially in absence of agent specialization, i.e., in the case of U-NCA, it is easier for the prototype individual to learn to pass a single message to all its clone neighbors and correctly interpret the information received. In addition, we highlight that U$\not D$-NCA are less prone to triggering vibrating dynamics by design, and are thus more successful in combination with SNNs, which display and take advantage of the same trait.

Concerning the importance of the neural model, we note that SNNs, either with or without homeostasis, surpass MLPs in 10 out of 12 cases. To better appraise the influence of homeostasis in SNNs, we need to narrow our focus to the re-assessment results, where this neural model leads to neatly superior outcomes in all but one case, confirming its fundamental role in self-regulation and adaptation. Moreover, we can re-state that SNNs seem to be more naturally suited for being combined with U$\not D$-NCA, as both tend to move away from high frequency non-adaptable behaviors. Therefore, we can conclude that the superiority of our contribution lies in the successful combination of the novel U$\not D$-NCA architecture with SNNs with homeostasis.

5 Concluding remarks

We explored the paradigm of collective control of Voxel-based Soft Robots (VSRs), a form of simulated modular soft robots, appraising the emergence of coordination from the synergistic actuation of individual agents, i.e., the voxels constituting the VSR. Taking inspiration from NCA, a form of distributed neural control, and from state-of-the-art forms of embodied control of VSRs, we introduced the novel concept of embodied Spiking Neural Cellular Automata (SNCA), in which we used Spiking Neural Networks (SNNs) as elementary units. To evaluate the performance of the proposed SNCA as a robotic controller, we compared it against the state-of-the-art embodied controllers, optimizing the controller parameters of three different VSRs for the task of locomotion. Our experimental results show that the SNCA is not only competitive with the pre-existing controllers, but it also leads to significantly more adaptable agents, outperforming their rivals when faced with unforeseen circumstances. Moreover, we highlight a trend towards less reality-gap prone behaviors in VSRs controlled by SNCA, which paves the way for the physical practicability of such robots.

We believe our contribution can be considered as a starting point for several additional analyses, spanning across diverse research directions. Concerning SNNs, we plan to experiment with neuroplasticity in the form of unsupervised learning, to the extent of achieving greater generality and increased robustness of controllers (Qiu et al., 2020). In addition, we will address the problem of collective control with a cooperative coevolution strategy aimed at optimizing an ensemble of heterogeneous SNCA controllers (Potter & Jong, 2000).

REFERENCES

Reem J Alattas, Sarosh Patel, and Tarek M Sobh. Evolutionary modular robotics: Survey and analysis. *Journal of Intelligent & Robotic Systems*, 95(3):815–828, 2019.

Peter Blouw, Xuan Choo, Eric Hunsberger, and Chris Eliasmith. Benchmarking keyword spotting efficiency on neuromorphic hardware. In *Proceedings of the 7th Annual Neuro-inspired Computational Elements Workshop*, pp. 1–8, 2019.

Maxence Bouvier, Alexandre Valentian, Thomas Mesquida, Francois Rummens, Marina Reyboz, Elisa Vianello, and Edith Beigne. Spiking neural networks hardware implementations and challenges: A survey. *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, 15(2): 1–35, 2019.

Nicholas Cheney, Jeff Clune, and Hod Lipson. Evolved electrophysiological soft robots. In *ALIFE 14: The Fourteenth International Conference on the Synthesis and Simulation of Living Systems*, pp. 222–229. MIT Press, 2014.

Francesco Corucci, Nick Cheney, Francesco Giorgio-Serchi, Josh Bongard, and Cecilia Laschi. Evolving soft locomotion in aquatic and terrestrial environments: effects of material properties and environmental transitions. *Soft robotics*, 5(4):475–495, 2018.

Andrea Ferigo, Giovanni Iacca, Eric Medvet, and Federico Pigozzi. Evolving Hebbian Learning Rules in Voxel-based Soft Robots. 2021.

Wulfram Gerstner and Werner M Kistler. *Spiking neuron models: Single neurons, populations, plasticity*. Cambridge university press, 2002.

Heiko Hamann. Swarm robotics: A formal approach. 2018.

Jonathan Hiller and Hod Lipson. Automatic design and manufacture of soft robots. *IEEE Transactions on Robotics*, 28(2):457–466, 2012.

Kazuya Horibe, Kathryn Walker, and Sebastian Risi. Regenerating Soft Robots Through Neural Cellular Automata. In *EuroGP*, pp. 36–50, 2021.

Wenlong Huang, Igor Mordatch, and Deepak Pathak. One policy to control them all: Shared modular policies for agent-agnostic control. In *International Conference on Machine Learning*, pp. 4455–4464. PMLR, 2020.

Eugene M Izhikevich. Which model to use for cortical spiking neurons? *IEEE transactions on neural networks*, 15(5):1063–1070, 2004.

Sam Kriegman, Nick Cheney, and Josh Bongard. How morphological development can guide evolution. *Scientific reports*, 8(1):1–10, 2018.

Xia Li and Anthony Gar-On Yeh. Neural-network-based cellular automata for simulating multiple land use changes using GIS. *International Journal of Geographical Information Science*, 16(4): 323–343, 2002.

Yi Li, Yingpeng Zhong, Jinjian Zhang, Lei Xu, Qing Wang, Huajun Sun, Hao Tong, Xiaoming Cheng, and Xiangshui Miao. Activity-dependent synaptic plasticity of a chalcogenide electronic synapse for neuromorphic systems. *Scientific reports*, 4(1):1–7, 2014.

Dirk M Lorenz, Alice Jeng, and Michael W Deem. The emergence of modularity in biological systems. *Physics of life reviews*, 8(2):129–160, 2011.

Wolfgang Maass. Networks of spiking neurons: the third generation of neural network models. *Neural networks*, 10(9):1659–1671, 1997.

Eric Medvet, Alberto Bartoli, Andrea De Lorenzo, and Giulio Fidel. Evolution of distributed neural controllers for voxel-based soft robots. In *Proceedings of the 2020 Genetic and Evolutionary Computation Conference*, pp. 112–120, 2020a.

Eric Medvet, Alberto Bartoli, Andrea De Lorenzo, and Stefano Seriani. 2D-VSR-Sim: A simulation tool for the optimization of 2-D voxel-based soft robots. *SoftwareX*, 12:100573, 2020b.

Eric Medvet, Alberto Bartoli, Andrea De Lorenzo, and Stefano Seriani. Design, validation, and case studies of 2D-VSR-Sim, an optimization-friendly simulator of 2-D Voxel-based Soft Robots. *arXiv preprint arXiv:2001.08617*, 2020c.

Eric Medvet, Alberto Bartoli, Federico Pigozzi, and Marco Rochelli. Biodiversity in evolved voxel-based soft robots. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 129–137, 2021.

Alexander Mordvintsev, Ettore Randazzo, Eyvind Niklasson, and Michael Levin. Growing neural cellular automata. *Distill*, 5(2):e23, 2020.

Giorgia Nadizar, Eric Medvet, Felice Andrea Pellegrino, Marco Zullich, and Stefano Nichele. On the effects of pruning on evolved neural controllers for soft robots. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pp. 1744–1752, 2021.

Giorgia Nadizar, Eric Medvet, Hola Huse Ramstad, Stefano Nichele, Felice Andrea Pellegrino, and Marco Zullich. Merging pruning and neuroevolution: towards robust and efficient controllers for modular soft robots. *The Knowledge Engineering Review*, 37, 2022.

Stefano Nichele, Mathias Berild Ose, Sebastian Risi, and Gunnar Tufte. CA-NEAT: evolved compositional pattern producing networks for cellular automata morphogenesis and replication. *IEEE Transactions on Cognitive and Developmental Systems*, 10(3):687–700, 2017.

Eyvind Niklasson, Alexander Mordvintsev, Ettore Randazzo, and Michael Levin. Self-Organising Textures. *Distill*, 6(2):e00027–003, 2021.

Deepak Pathak, Christopher Lu, Trevor Darrell, Phillip Isola, and Alexei A Efros. Learning to control self-assembling morphologies: a study of generalization via modularity. *Advances in Neural Information Processing Systems*, 32, 2019.

Sidney Pontes-Filho and Stefano Nichele. A Conceptual Bio-Inspired Framework for the Evolution of Artificial General Intelligence. *arXiv preprint arXiv:1903.10410*, 2019.

Mitchell A Potter and Kenneth A De Jong. Cooperative coevolution: An architecture for evolving coadapted subcomponents. *Evolutionary computation*, 8(1):1–29, 2000.

Huanneng Qiu, Matthew Garratt, David Howard, and Sreenatha Anavatti. Towards crossing the reality gap with evolved plastic neurocontrollers. In *Proceedings of the 2020 Genetic and Evolutionary Computation Conference*, pp. 130–138, 2020.

Michael Rubenstein, Alejandro Cornejo, and Radhika Nagpal. Programmable self-assembly in a thousand-robot swarm. *Science*, 345(6198):795–799, 2014.

Erica Salvato, Gianfranco Fenu, Eric Medvet, and Felice Andrea Pellegrino. Crossing the Reality Gap: a Survey on Sim-to-Real Transferability of Robot Controllers in Reinforcement Learning. *IEEE Access*, 2021.

Shyam Sudhakaran, Djordje Grbic, Siyan Li, Adam Katona, Elias Najarro, Claire Glanois, and Sebastian Risi. Growing 3D Artefacts and Functional Machines with Neural Cellular Automata. *arXiv preprint arXiv:2103.08737*, 2021.

Jacopo Talamini, Eric Medvet, Alberto Bartoli, and Andrea De Lorenzo. Evolutionary synthesis of sensing controllers for voxel-based soft robots. In *Artificial Life Conference Proceedings*, pp. 574–581. MIT Press, 2019.

Gina G Turrigiano and Sacha B Nelson. Homeostatic plasticity in the developing nervous system. *Nature reviews neuroscience*, 5(2):97–107, 2004.

Fuda van Diggelen, Eliseo Ferrante, Nihed Harrak, Jie Luo, Daan Zeeuwe, and AE Eiben. The Influence of Robot Traits and Evolutionary Dynamics on the Reality Gap. *IEEE Transactions on Cognitive and Developmental Systems*, 2021.

Alexandre Variengien, Sidney Pontes-Filho, Tom Glover, and Stefano Nichele. Towards self-organized control: Using neural cellular automata to robustly control a cart-pole agent. *Innovations in Machine Intelligence*, 1, 2021.

Xiuqing Wang, Zeng-Guang Hou, Anmin Zou, Min Tan, and Long Cheng. A behavior controller based on spiking neural networks for mobile robots. *Neurocomputing*, 71(4-6):655–666, 2008.

ISBN 978-82-326-5746-9 (printed ver.)
ISBN 978-82-326-6953-0 (electronic ver.)
ISSN 1503-8181 (printed ver.)
ISSN 2703-8084 (online ver.)