INNOVATIONS IN MACHINE INTELLIGENCE
NATURAL PRINCIPLES AND ARTIFICIAL MODELS

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

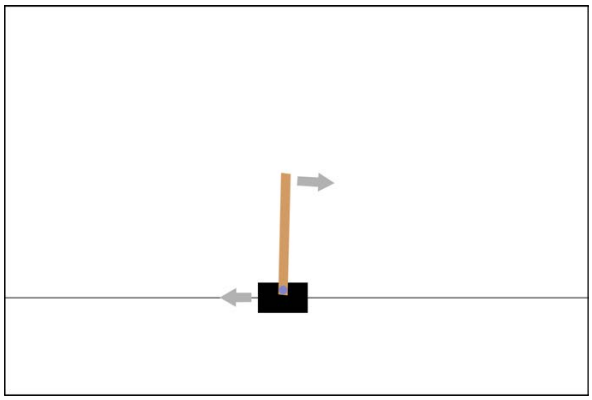

Figure 1. The cart-pole environment. The top arrow represents the angular velocity of the pole, the bottom arrow, the velocity of the cart.

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.

$$r_t = \begin{cases} \cos\left(\frac{x\pi}{2L}\right)^2 & \text{if } t \text{ is not a final step} \\ -100 & \text{else.} \end{cases} \quad (1)$$

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/

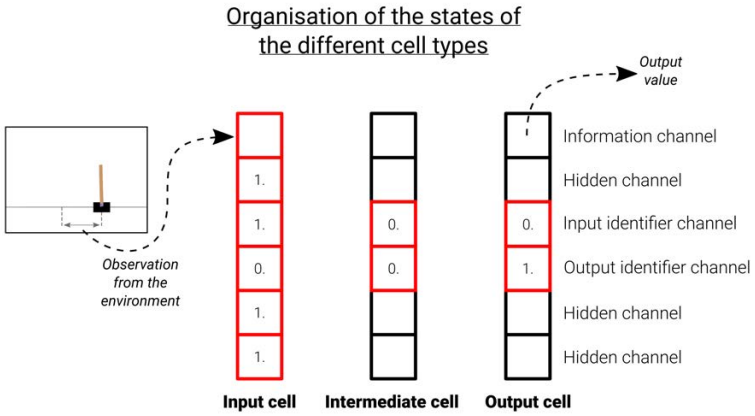

Figure 2. The role of the different channels and types of cells. Each cell is composed of a 6-dimensional vector, each dimension is called a channel. Different channels have different behaviors: some are fixed (in red) others can change according to the update function (in black). Moreover, channels 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.

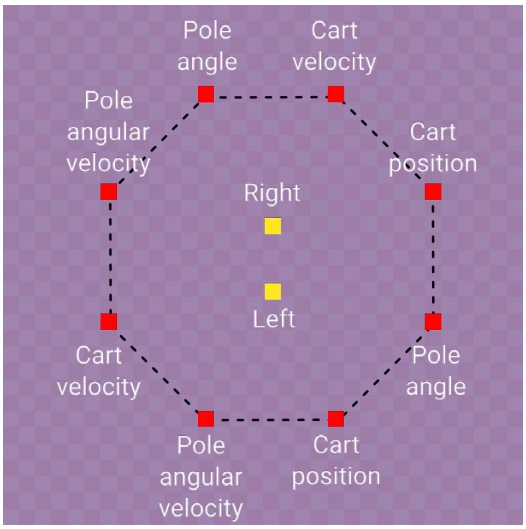

Figure 3. The position of input (in red) and output cells (in yellow).

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.

$$Q(s_t, a_t) = E[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + ... | s_t, a_t] \quad (2)$$

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].

$$\text{Loss} = \text{L2}(outputs, target) + \frac{\lambda}{N^2} \sum_{i,j=1}^{N} \sum_{chan=1}^{6} (\text{clip}(Grid_{i,j,chan}, -5, 5) - Grid_{i,j,chan})^2 \quad (3)$$

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.

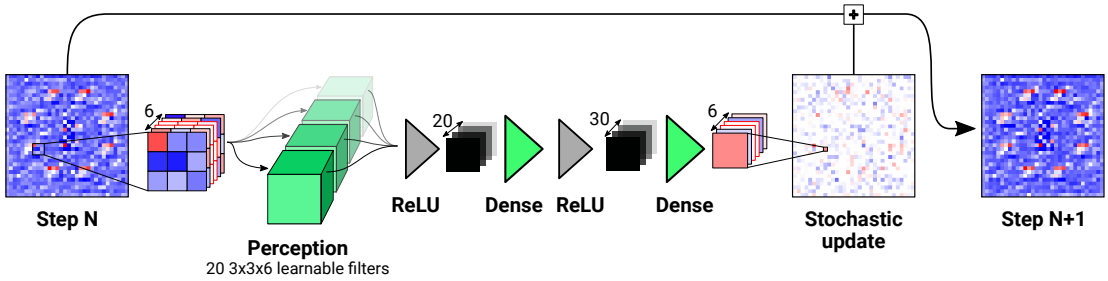

Figure 4. The architecture of our model. Positive values are depicted in red, negative in blue. Green objects identify the learnable parameters of the model.

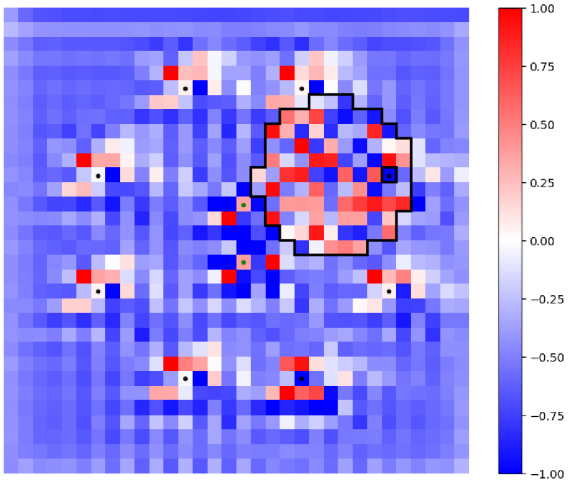

Figure 5. A damaged grid is shown on the information channel. The shape of the damaged region is an irregular circle due to rasterization effects.

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.

- $$y_j = \begin{cases} r_j & \text{if } j \text{ is a final step} \\ r_j + \gamma * \max_{a'} Q(o_{j+1}, a'; \theta) & \text{else.} \end{cases} \quad (4)$$
- $$TaskLoss = (y_j - Q(o_j, a_j; \theta))^2 \quad (5)$$

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

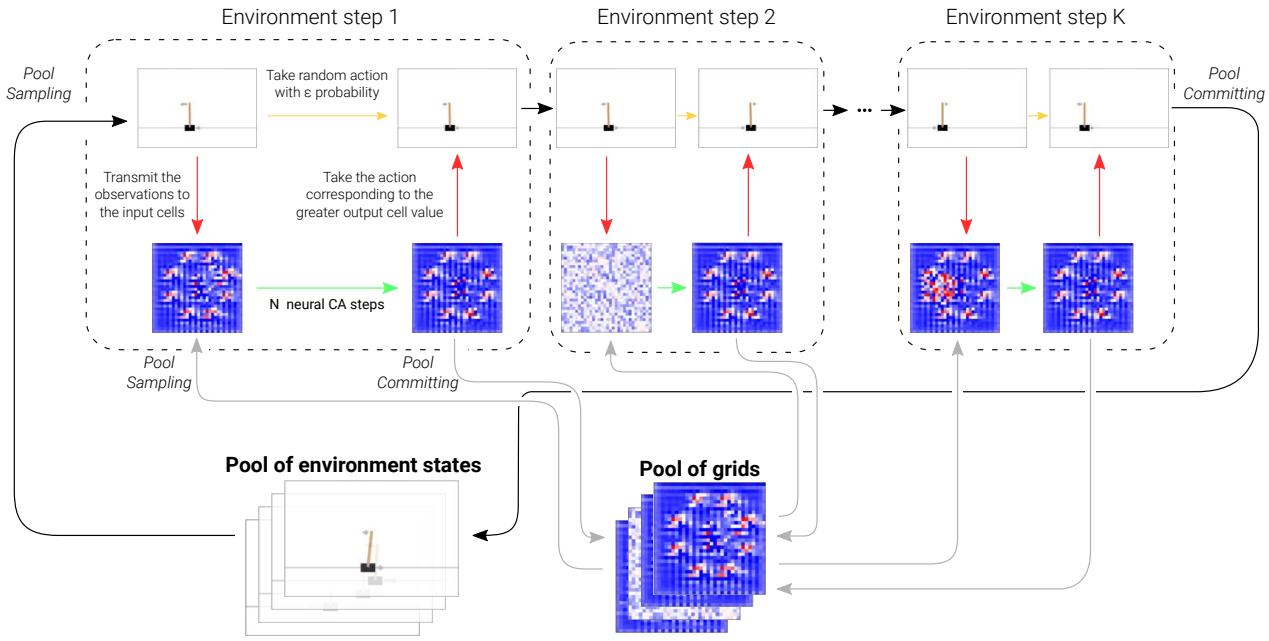

Figure 6. The procedure for the exploration phase. We match cart-pole states and grids randomly sampled from two independent pools. This provides a way to simulate long-term dynamics both for the cart-pole environment and for the neural CA. In practice, we used K=2.

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/.

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

3 https://colab.research.google.com/github/aVariengien/self-organized-control/blob/main/code/Towards-self-organized-control-notebook.ipynb

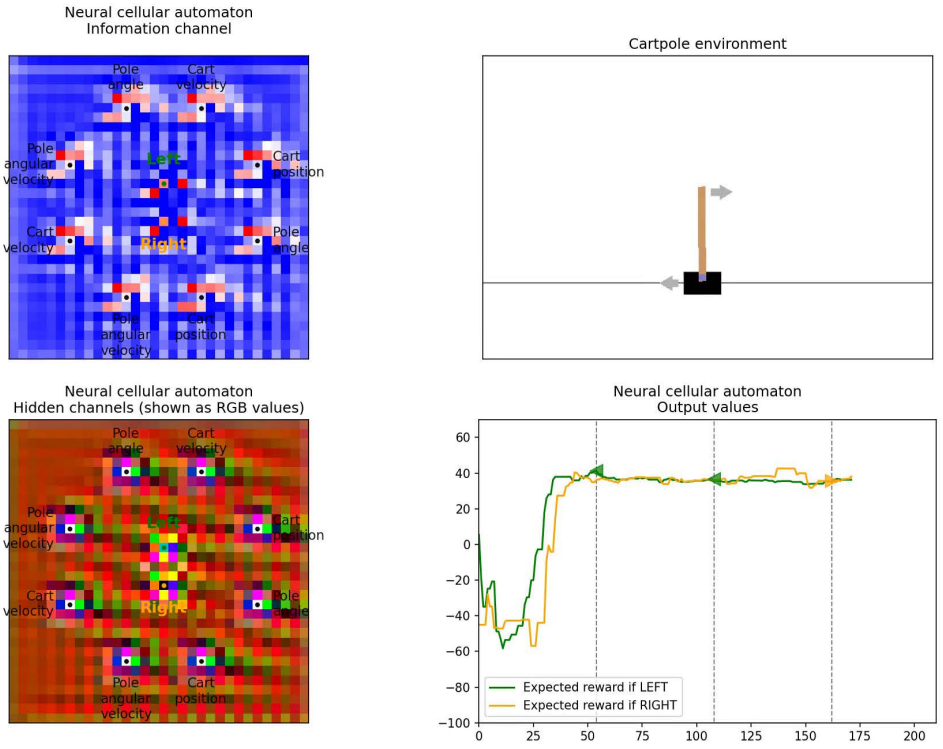

Figure 7. On the left, the states of the neural CA. For the information channel, negative values are in blue, positive in red while the hidden channel is represented as RGB values. Bottom right: the plot of the output values of the neural CA. The vertical dotted lines denote when the action that has 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/

.

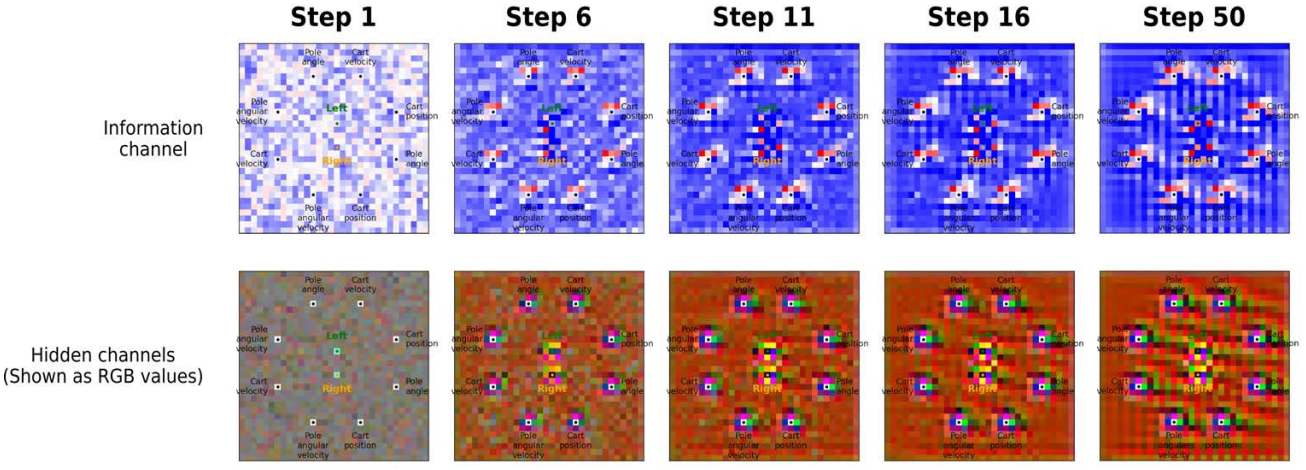

Figure 8. The state of the grid during the 50 first phase. The information channel is displayed on the top. Negative values are in blue, positive in red while the hidden channels (on the bottom) are represented as RGB values. We observe an evolution of the grid from a disorganized state to a precise, 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*.

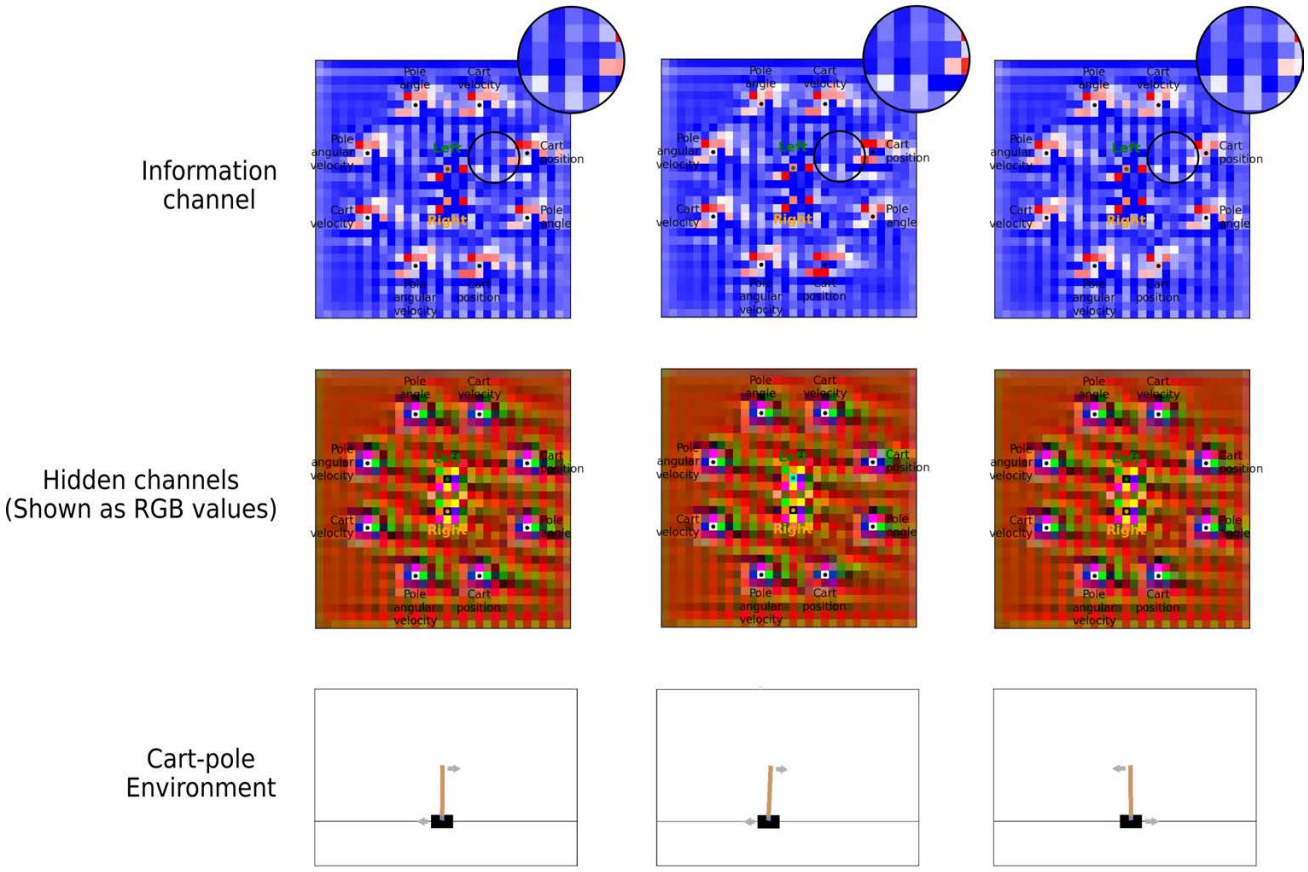

Figure 9. The state of the grid and of the environment in 3 situations, taken from the same run. The information channel is displayed on the top. Negative values are in blue, positive in red, while the hidden channels (on the bottom) are represented as RGB values. We observe that the global pattern of 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.

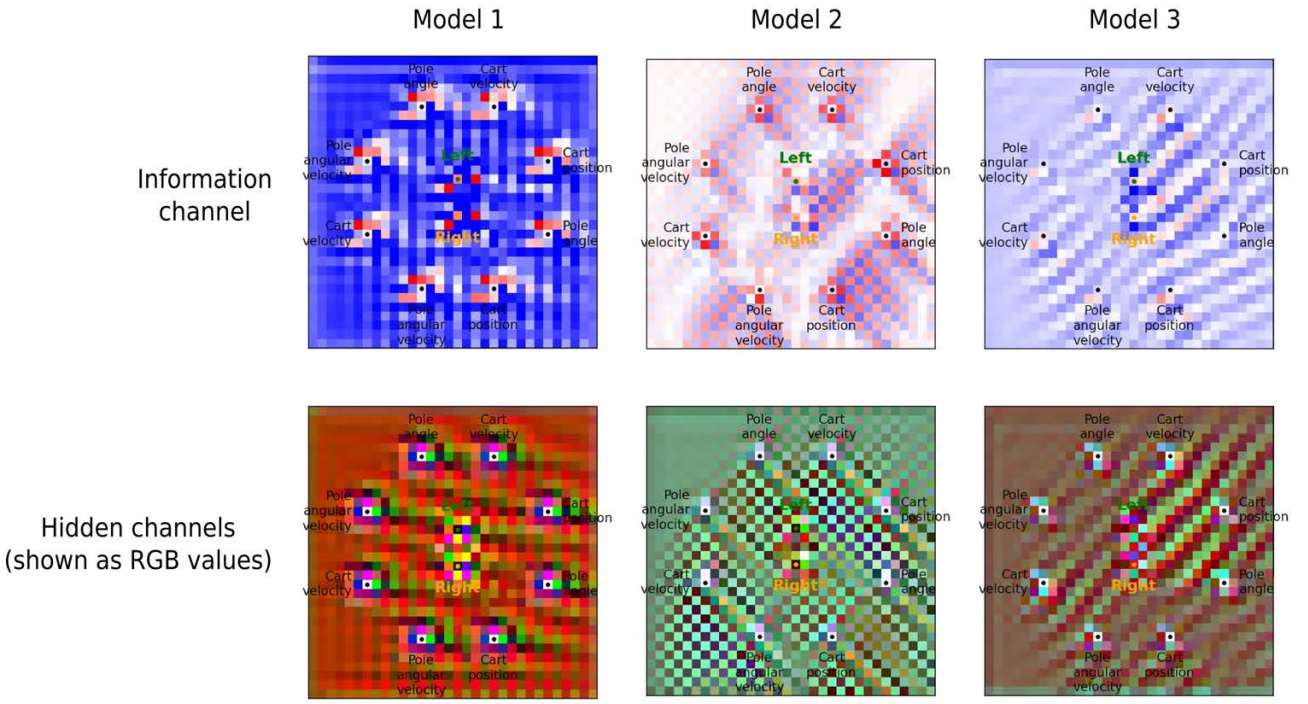

Figure 10. The stable spatial organization of the grid for 3 independently trained neural CA. The information channel is displayed on the top. Negative values are in blue, positive in red, while the hidden channels (on the bottom) are represented as RGB values. Their dynamics can be observed in videos available at https://avariengien.github.io/self-organized-control/

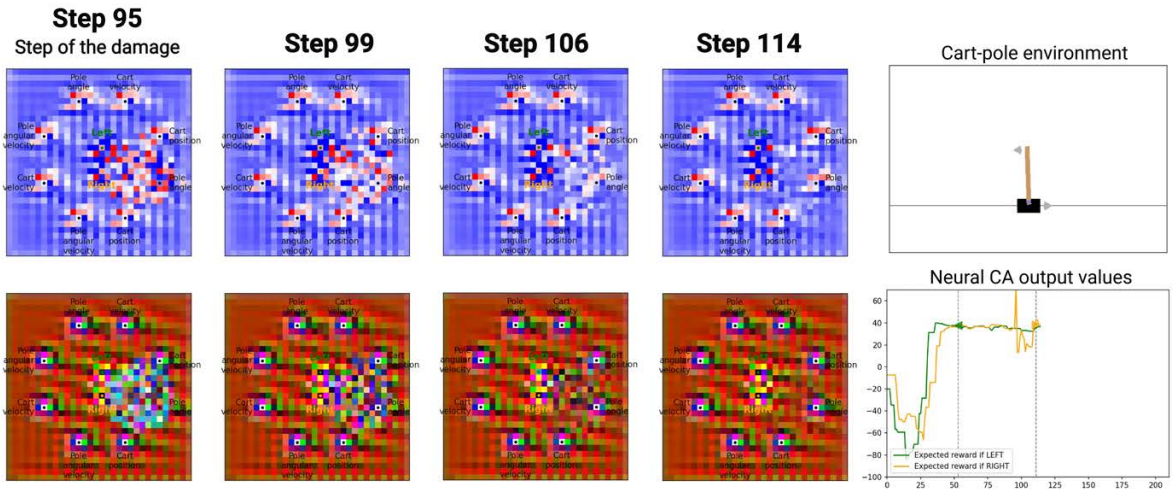

Figure 11. A slide sequence of grids during the recovery from damage. The information channel is displayed on the top. Negative values are in blue, positive in red, while the hidden channels (on the bottom) are represented as RGB values. On the left is the state of the cart-pole during the damage and 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.

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

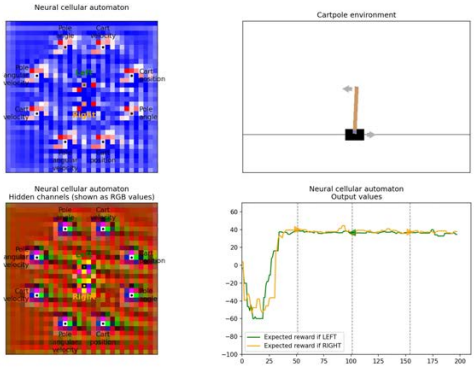

Figure 12. A neural CA controls a cart-pole agent with noisy updates. On the left is the grid of the neural CA. For the information channel, negative values are in blue, positive in red while the hidden channel is represented as RGB values. Bottom right: the plot of the output values of the neural CA. 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.

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.

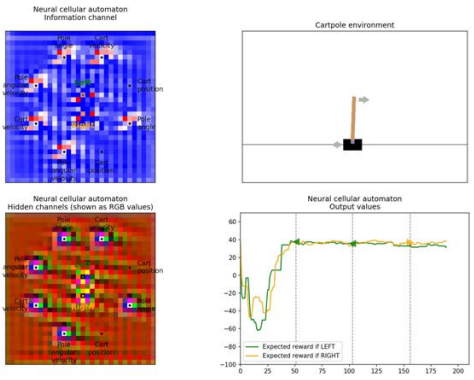

Figure 13. A neural CA controls a cart-pole agent with two deleted input cells. On the left, the neural CA grid. The input cells corresponding to the cart position have been deleted and replaced by intermediate cells. No noise nor damage was added. On the left is the grid of the neural CA. For the information 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]

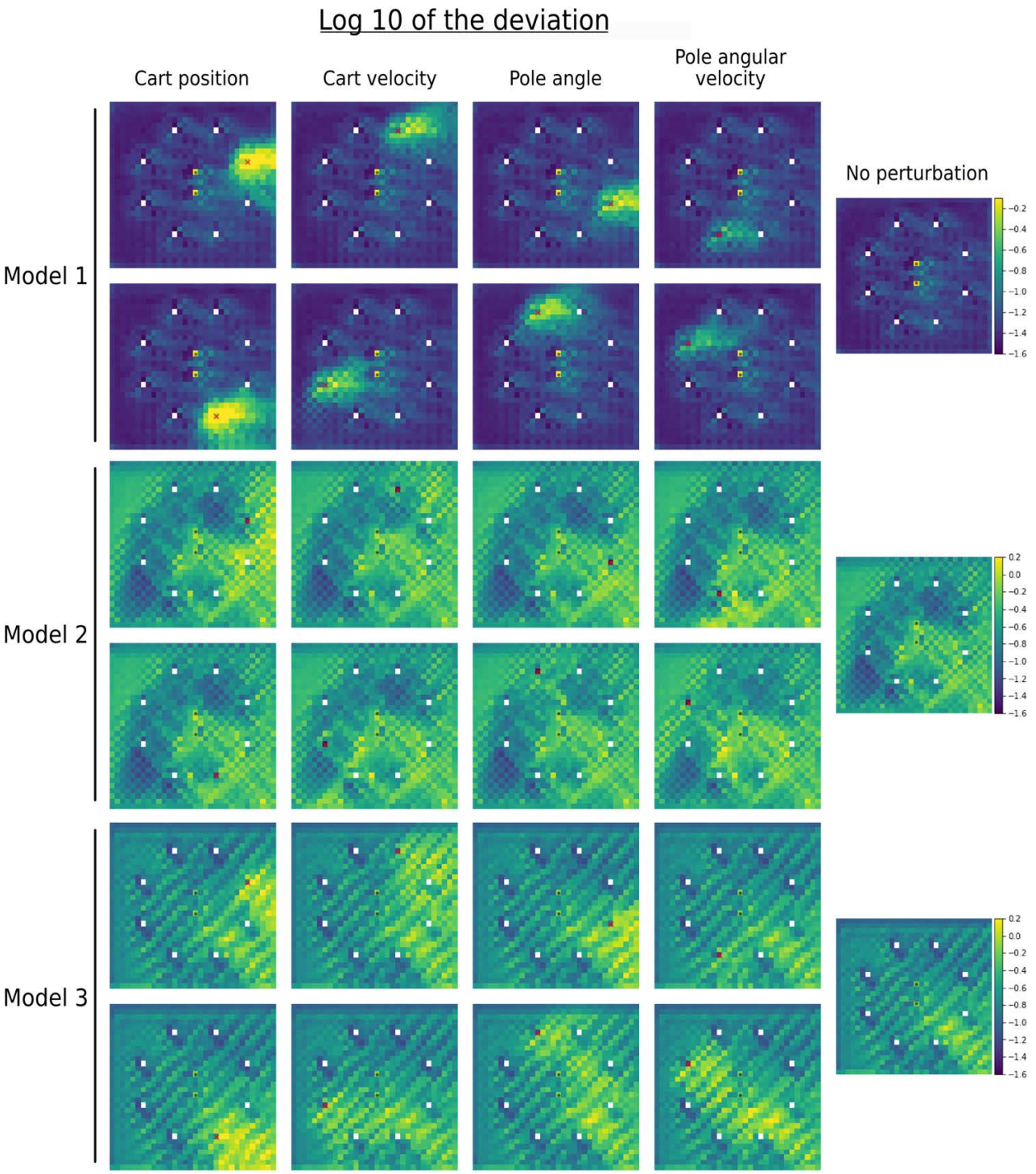

Figure 14. Visualization of the region of influence of each input cell. We plotted the decimal logarithm of the deviation between a standard input and a perturbed one. The perturbed input cell is symbolized by a red cross, black dots identify the output cells. The natural deviation without any perturbation, 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 ϵ 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
      $y_k \leftarrow \begin{cases} r_{t_k} & \text{if } t_k \text{ is a final step} \\ r_{t_k} + \gamma * \max_a Grid_{k,x_a,y_a,0} & \text{else.} \end{cases}$
      for *step* = 1, RandomInt($Step_{min}$, $Step_{max}$) **do**
        $Grid'_k \leftarrow$ NCA($Grid'_k$, o_{t_k})
      end for
      $TaskLoss_k \leftarrow (Grid'_{k,x_{a_{t_k}},y_{a_{t_k}}} - y_k)^2$
      $Loss_k \leftarrow TaskLoss_k + \lambda * \text{OverFlow}(Grid'_k)$
    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.