Adversarial Takeover of Neural Cellular Automata

Lorenzo Cavuoti[1], Francesco Sacco[2], Ettore Randazzo[3] and Michael Levin[4]

[1]University of Trieste, [2]University of Pisa, [3]Google Research, [4]Allen Discovery Center
lorenzocav97@gmail.com, francesco215@live.it

Abstract

The biggest open problems in the life sciences concern the algorithms by which competent subunits (cells) could cooperate to form large-scale structures with new, system-level properties. In synthetic bioengineering, multiple cells of diverse origin can be included in chimeric constructs. To facilitate progress in this field, we sought an understanding of multi-scale decision-making by diverse subunits beyond those observed in frozen accidents of biological phylogeny: abstract models of life-as-it-can-be. Neural Cellular Automata (NCA) are a very good inspiration for understanding current and possible living organisms: researchers managed to create NCA that are able to converge to any morphology. In order to simulate a more dynamic situation, we took the NCA model and generalized it to consider multiple NCA rules. We then used this generalized model to change the behavior of a NCA by injecting other types of cells (adversaries) and letting them take over the entire organism to solve a different task. Next we demonstrate that it is possible to stop aging in an existing NCA by injecting adversaries that follow a different rule. Finally, we quantify a distance between NCAs and develop a procedure that allows us to find adversaries close to the original cells.

Introduction

Biology operates in a multiscale competency architecture: cells follow local rules in ways that result in interesting and robust large-scale patterns. Major knowledge gaps, despite progress in molecular genetics, include the policies guiding individual cell behaviors toward body-level anatomical structures. This especially concerns the algorithms needed to reliably reach a consistent form under a range of changing conditions. Cellular behavior is guided in part by gene-regulatory networks inside cells, and coordinated by biochemical and bioelectrical networks at the tissue level. Both of these can be represented as neural networks that guide the mechanisms determining the cell's activity Moore et al. (2018); Biswas S (2016).

Neural Cellular Automata (NCA) represent a recent development of Cellular Automata, where the underlying rule is represented as a neural network and is learned using gradient-based optimization Mordvintsev et al. (2020); the NCA starts from a seed state and is trained to reach a target state (figure 1).

Figure 1: Example of seed and target state, left: the seed state, right: the target state.

In recent advances in this field, scientists have managed to change the global properties of a NCA by adding some cells that follow a different rule Randazzo et al. (2021); this corresponds to biological situations when cells of diverse genetics are assembled into chimeras Nanos V (2021). However these cells remain fixed and can't expand in space.

This is a problem, because the number of new cells (aka. adversaries) required must be relatively high in order to steer the behavior of the whole organism, however, substituting a high number of cells in a biological organism could be difficult. In general, one task in biomedical interventions and in guided self-assembly (bioengineering) contexts is to find the minimal intervention that achieves a given outcome.

One way to minimize the intervention is to generalize NCA to multiple rules, allowing us to simulate what happens when one type of cell overtakes the other. This way we can inject very few adversaries and let them take over other types of cells (figure 2). Furthermore, this model is a generalization of the NCA model[1] that is more biologically plausible, since it can simulate the growth of a group of cells at the expense of another.

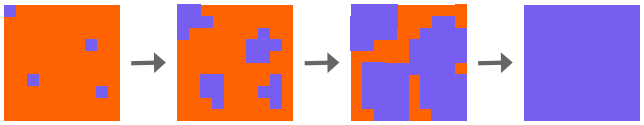

Figure 2: In orange we indicate the original cells while in blue the adversarial cells. The adversaries take over the original cells and change the behavior of the whole organism.

[1]Because, if all the rules are the same, the model behaves like evolving with a single rule.

We then apply the model to two different scenarios:

- **Changing static properties** of a NCA: Examples of such properties are changing the color of an organism (figure 3), adding or removing a limb, making the tail longer, and so on. All of these properties can be observed with a before/after photo of the organism.
- **Changing dynamical properties** of a NCA: Examples of these properties are altering the lifespan of an organism, or the ability to regenerate damage. These properties are much more interesting from an aging and regenerative medicine point of view, however, as we will see, they are much harder to train than the static ones.

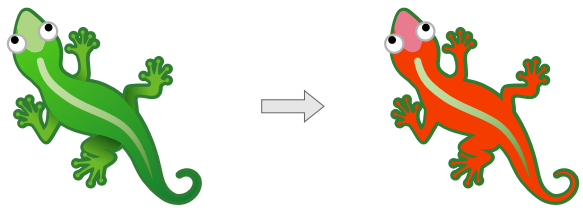

Figure 3: Example of changing a static property, the lizard color turns from green to red.

Lastly, the parameters of the adversaries can become drastically different from the original cells they replace, which presents biologists with the challenge of identifying DNA or pharmacological reagents that change some of the cells' behaviors in the necessary fashion. This gives rise to an important inverse problem Lobo D (2014): what can be tweaked at the lowest level (e.g., DNA mutations) to give rise to desired changes at the system level (anatomy)? The difficulty of solving this problem is what prevents true Lamarckian inheritance, and also limits regenerative medicine applications of modern technologies such as CRISPR.

Therefore, we explore ways to make the parameters of the adversarial cells as similar as possible to the original cells, while still being able to accomplish the given task, demonstrating that only a small change in the parameters is sufficient to turn an original cell into an adversarial one.

Figures in video form and the code is available here[2].

The model

NCA model

Before diving into the generalization of NCA, we summarize how the NCA model works; a better explanation can be found in the original paper Mordvintsev et al. (2020).

A Cellular Automata (CA) consists of a grid of cells that is iteratively updated using the same update rule at each step Neumann and Burks (1966), the only requirement is that the next state of each cell depends only on its previous state, x_t and the state of its neighbors, $N(x_t)$.

$$x_{t+1} = f(x_t, N(x_t))$$

Neural Cellular Automata (NCA) use a neural network to model the function f and consider the states x_t to be continuous, this allows training f using gradient-based optimization. The cell state x_t is represented by a vector where the first 4 components represent the RGBA channels of the pixel and the remaining are hidden channels that allow the NCA to pass information between its cells (figure 5 left).

The α channel (transparency) has an important role: if a cell has $\alpha > 0.1$ it means that the cell is mature, otherwise it's dead. This distinction is essential because a cell can change its state if at least one of its immediate neighbors[3], or itself, is mature (figure 4), if this is not the case its state is set to 0. The evolution starts with only one mature cell in the center of the canvas, then the cells are evolved and reach the target image (figure 1).

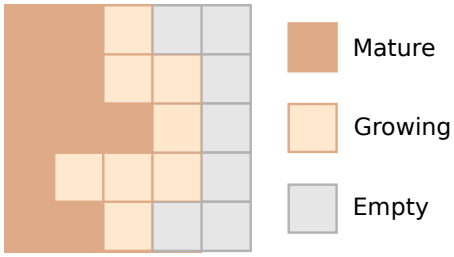

Figure 4: Illustration of mature, growing and empty cells, growing cells are in the immediate neighborhood of mature ones. Image adapted from Mordvintsev et al. (2020), licensed under CC BY 4.0.

Multiple NCA model

We call our new model Multiple NCA because it generalizes a single NCA to multiple update rules. For ease of explanation, we are going to consider the case of only 2 rules, f_1 and f_2.

Masking Since the α channel tells whether a cell is alive or dead, if we have two different types of cells we need two alpha channels, α_1 and α_2. In this new model we decided to put the alpha channels at the end of the state vector (figure 5).

[2] https://letteraunica.github.io/neural_cellular_automata/extra

[3] We consider a Moore neighborhood.

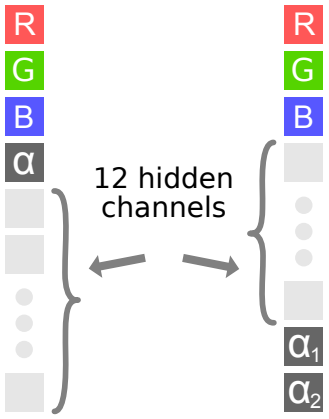

Figure 5: left: location of the channels in the NCA model, right: channels location in the Multiple NCA model.

To make the model more realistic we added some constraints:

1. A cell can be mature in only one channel, this means that no cell can have both alphas > 0.1. We do this because we consider the two cells as having different DNA, so they must have different rules and there is no in-between.

2. We impose that new cells can only grow near mature ones of the same type, example: cells of type 2 can only grow near mature cells of type 2.

3. When both alphas are in $0 \leq \alpha < 0.1$, and the cell is near a mature one of both f_1 and f_2, the cell evolves following the average of both rules (figure 6). Biologically this means that two kinds of cells are fighting for the control of one spot. Mathematically, we did this because otherwise we would have a privileged rule. This also implies that, if the two rules are the same, the system acts identically to what it did if it was evolved with only one rule.

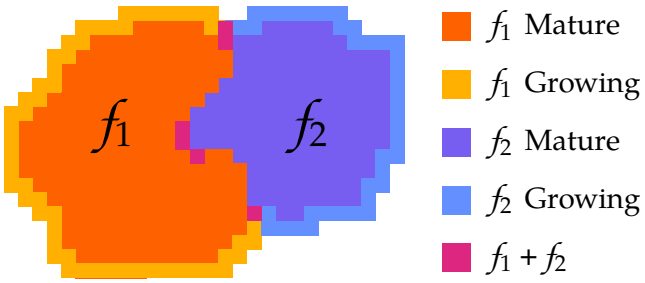

Figure 6: Representation of two different rules evolving toghether in the Multiple NCA model. Magenta cells are growing cells of both f_1 and f_2, so they evolve following the average of both rules.

Furthermore, we change the perception stage and the output of the NCA as follows.

Perception The Multiple NCA model perceives only the sum of all alpha channels[4], and not the individual channels, we do it for two main reasons:

- Computational: we have 2 alpha channels, however, the NCA model uses only one alpha channel, which means that we need to find a function that reduces the number of alpha channels before passing the state to the NCA model to be evolved; summing up the two alpha is one of the simplest ways to do this without having a privileged rule.
- Biological: Summing up the two alpha has a nice biological interpretation, it assumes that a cell is aware of its surroundings but can't distinguish the type of the neighboring cells trivially, since it doesn't have access to α_1 and α_2; this encourages a NCA to take advantage of the hidden channels to be able to distinguish itself from the other NCAs.

From an implementation standpoint we only need a function that takes a state x, sums the alpha, and places the sum right after the RGB components (figure 7); this new state can now be passed to the NCA to be evolved.

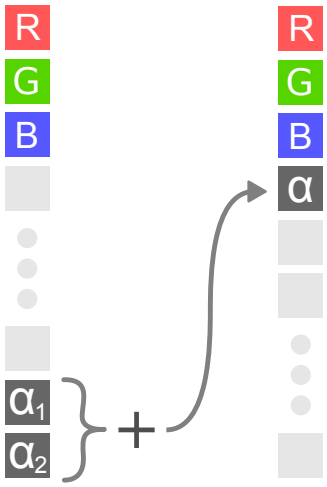

Figure 7: Before passing the state to f_1 or f_2 to be evolved, we sum up α_1 and α_2 then place the sum in the right location in the state vector.

Output We impose that each NCA can only update its alpha channel and not the other ones (figure 8). This makes sense because we don't want a cell of type 1 to edit the alpha channel of a cell of type 2 and therefore kill it trivially.

[4]Other possible functions could have been:

1. Weighted sum of the alphas. However, this implies that there is a privileged rule.

2. Randomly choose an alpha channel to be perceived. However, this doesn't have a nice biological interpretation like summing up the alpha channels.

Instead, if the adversaries want to take over, they must rely on changing their internal state in such a way that makes the original cells undergo the process of apoptosis.

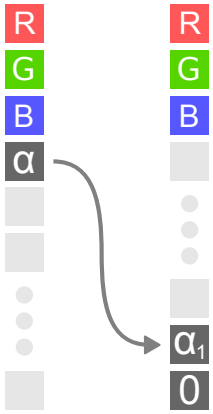

Figure 8: After f_1 computes the update we take α_1 and put it at the end of the state vector, leaving α_2 unchanged.

The whole update step can be seen in figure 9.

Training technique

One of the first issues we encountered, was that the adversarial cells never tried to overtake the original cells, so they never took over the organism. We solved this problem by penalizing the percentage of old cells still present, like so:

$$L = L_{target} + \lambda N_{old}$$

Where L_{target} is the distance to the target image, N_{old} is the number of old cells[5], and λ is a hyperparameter. Considering a loss function like this, nevertheless, leads to another problem: when we first introduce the new cells N_{old} is very high, which in turn makes the loss very high. This means that the adversaries will trade some of the image quality in favor of a faster cell replacement. A solution could be to give the NCA plenty of time before evaluating the loss, however, the NCA might learn to destroy the image at the start, just to rebuild it before the loss evaluation. To address both these problems we made a custom loss function that is dependent on the number of steps n.

$$L = \sum_{n=n_{start}}^{n_{end}} \lambda_1(n) L_{target}(n) + \lambda_2(n) N_{old}(n)$$

Now the hyperparameters λ_1 and λ_2 are functions that depend on the number of steps n, while L_{target} and N_{old} represent respectively the distance from the target image and the number of old cells at the n-th step.

[5]This is slightly wrong since N_{old} is not differentiable, so in practice we used the sum of the α_1 channel over the entire image. However, it is more intuitive to think about penalizing N_{old} rather than α_1.

Experiments

Changing static properties

As we stated in the introduction, static properties include changing the color of an organism, adding or removing a limb, making the tail longer, and so on. We decided to apply our model to 3 cases of increasing difficulty (figure 10):

1. Turning the lizard from green to red: the shape remains fixed but the color of the organism changes.
2. Removing the tail of the lizard: the shape of the organism changes but the color remains the same[6].
3. Turning a bug into a butterfly: both the shape and the color pattern change dramatically.

In every case we start with a pre-trained Persistent NCA (which means it reaches the final image and keeps it for an infinite amount of time) and inject a small percentage of adversaries in it, then we only train the adversaries in order to change the appearance of the entire organism.

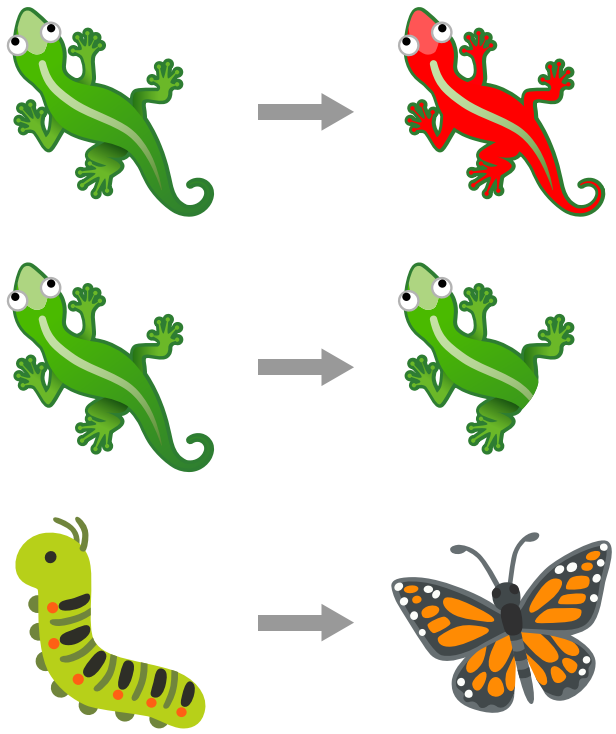

Figure 10: Illustration of the three static experiments.

[6]We found that altering the shape is harder because the adversaries have to learn to overtake the organism first, and, once the original cells are gone, they have to remove the tail.

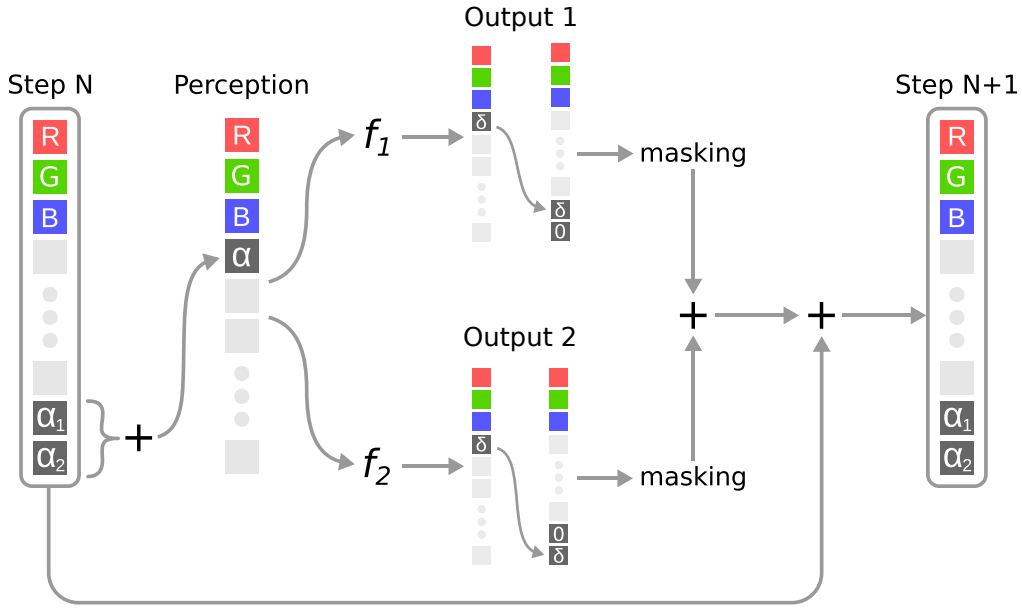

Figure 9: Illustration of the entire forward pass of the Multiple NCA model. f_1 and f_2 represent the two NCA rules, while masking refers to the 3 operations described in the Masking subsection.

Results In figure 12 we plotted the evolution of the NCAs, the time indicates the number of steps since the adversaries are first injected, we always inject the adversaries in a 2x2 square randomly located inside the organism. As you can see, the adversaries learn to influence the original cells to undergo the process of apoptosis, thus taking over the whole organism. Furthermore, as they are expanding, they try to match the color and target shape.

From a biological point of view, we think that adding the adversaries in a small square is more interesting than using, for example, a spray pattern (figure 11), because it shows that we can edit the cells in a single location then the change propagates throughout the body.

Figure 11: Examples of spray patterns with different percentages of adversaries, in blue, of respectively, $5\%, 25\%$ and 50%.

From a practical perspective, we noticed that training a model with adversaries that start from a small square is much harder than training starting from a spray pattern (figure 11), however, models that are able to take over starting from a small square generalize well to spray patterns, while the opposite isn't true, these results can be found here.

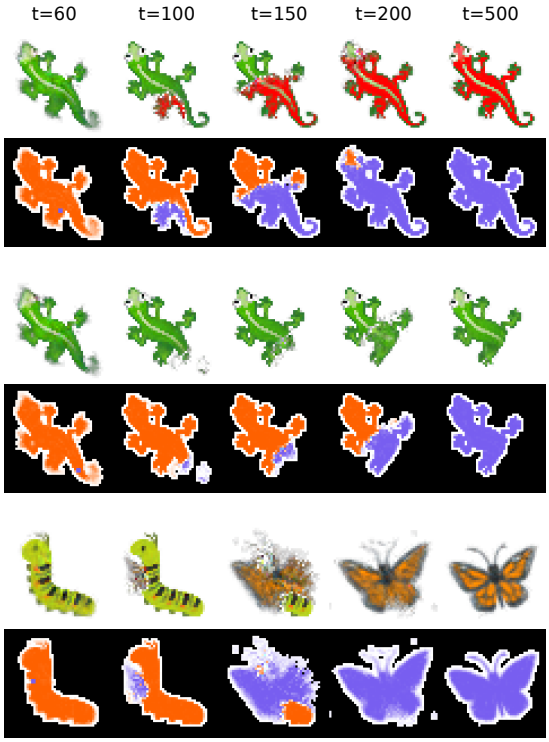

Figure 12: Results of the static experiments, first we plot the evolution of the organism, and right below it the cell mask, which tells where the original (orange) and the adversarial (blue) cells are located. t refers to the number of steps since the beginning of the evolution, we inject the adversaries always at step 60. We encourage the reader to take a look at the videos of this evolution at `https://letteraunica.github.io/neural_cellular_automata/extra#static-properties`

Changing dynamic properties

Changing a dynamic property means to change the whole NCA evolution, for example avoiding the decayment of a Growing NCA (figure 13). In this part we focus on turning 3 Growing NCA into Persistent ones, this is much harder than the previous experiment because the organism remains in the final state only for a limited amount of time, so the adversaries have to take over the whole organism before it decays.

Results In figure 13 we plot the evolution of 3 Growing NCA before we turn them into Persisting ones, in all cases we perform the adversarial injection at step t=60, which corresponds to the time the Growing NCA reaches the target state.

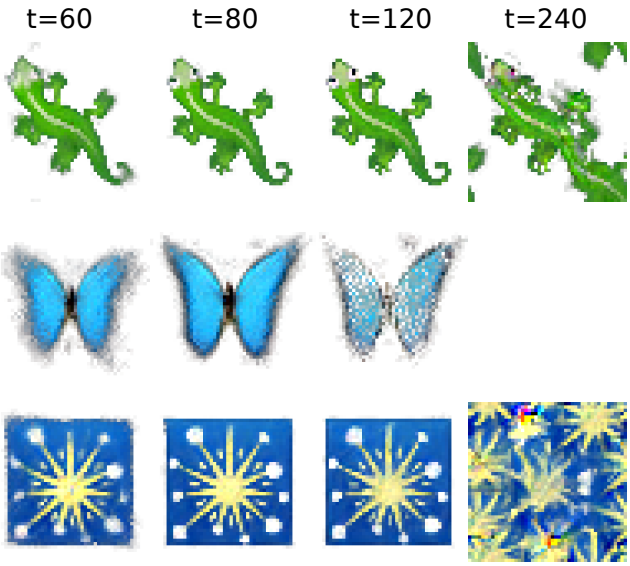

Figure 13: Growing NCA rules that we used in the dynamic experiment. After some steps they degenerate.

The Growing lizard was the easiest to turn into Persistent, because it decays at about step 200. This leaves about 140 steps for the adversaries to take over the organism, which are sufficient even when injecting the adversaries in a small 3x3 square seed (figure 14).

The butterfly was a little harder. In this case the organism decays by vanishing at about step 120. We see that the adversaries exploit this feature and take over the organism exactly when it vanishes. We weren't able to train this NCA with a square seed, which indicates the difficulty of the task. Still, we managed to reduce the initial number of adversaries to a very low number, only 3% of the total cells.

Finally, to turn a Growing NCA of the firework into a Persistent one, we needed about 50% of cells substituted, much higher than the lizard and the butterfly. We think this is due to 2 factors:

1. Unlike the butterfly, the firework decays by exploding rather than vanishing (figure 13).

2. Unlike the lizard, the firework decays at about step 140 (figure 13), since we perform the adversarial injection at step 60, this leaves only 80 steps for the adversaries to take over.

These two factors combined mean that the adversaries must take over before the growing cells explode. However, this time is very limited, which leads to a high initial number of adversaries.

From these experiments we can hypothesize that the more time it takes for the organism to decay the lower the initial percentage of adversaries is needed.

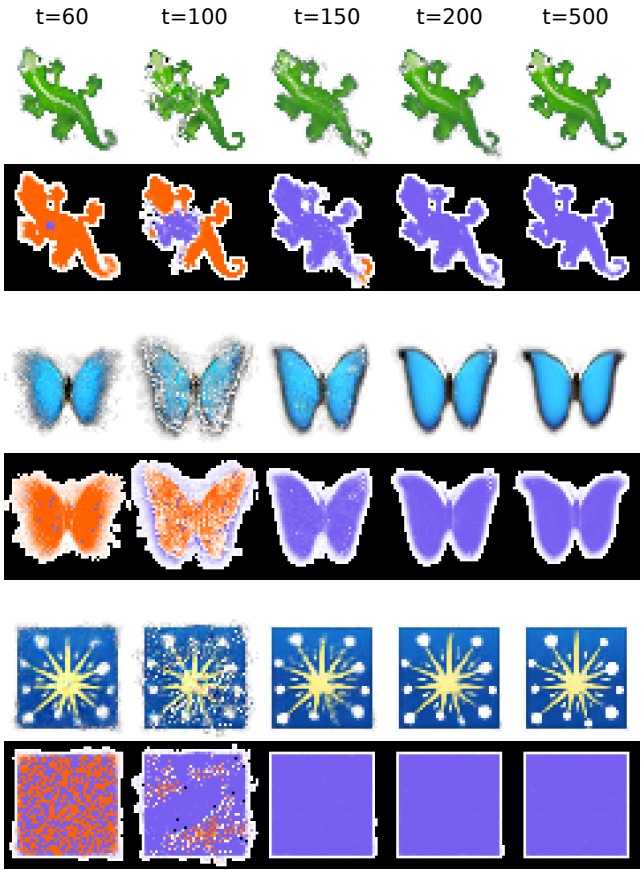

Figure 14: Results of the dynamic experiments, as before, first we plot the evolution of the organism, and right below it the cell mask, which tells where the original (orange) and the adversarial (blue) cells are located. t refers to the number of steps since the beginning of the evolution, we inject the adversaries always at step 60. We encourage the reader to take a look at the videos of this evolution at `https://letteraunica.github.io/neural_cellular_automata/extra#dynamic-properties`

Adding a perturbation

Iterated maps, like cellular automata and differential equations, oftentimes lead to chaotic systems. This implies that small changes to the initial conditions or to the function parameters, will lead to completely different results after some time Berto and Tagliabue (2022).

This is a double-edged sword:

- On one hand, chaotic systems, by definition, are very hard to predict and understand.
- On the other hand, lying at the edge of chaos gives us the power of influencing the system by a lot, with very little changes to its parameters Berto and Tagliabue (2022). Mother nature knows this very well, for example, humans have 99% of the DNA in common with chimpanzees, yet we are very different from them.

In the previous paragraphs, when training the adversaries, oftentimes the parameters become widely different from the ones of the original cells. This is a problem, because in a real organism we would like to edit the cells as little as possible. In this section we try to fix this problem by finding adversaries with parameters close to the original NCA parameters.

The model

As we said, we'd like to have adversaries with weights that are only a little perturbation off the original ones:

$$w_{new} = w_{old} + \Delta w$$

To be sure that the perturbation Δw remains as small as possible, we added an additional term in the loss, which penalizes the L^2 norm of the perturbation $|\Delta w|_2$, so the total loss will be:

$$L = L_{target} + \lambda_1 N_{old} + \lambda_2 |\Delta w|_2$$

Where L_{target} is the distance to the target image and λ_1, λ_2 are hyperparameters. We used two different metrics to evaluate the results, the norm of the perturbation $|\Delta w|_2$ and the the cosine similarity between w_{old} and w_{new}.

$$cos(w_{old}, w_{new}) = \frac{\langle w_{old}, w_{new} \rangle}{|w_{old}||w_{new}|}$$

Results

In figure 15 you can see the metrics for turning a green Persistent lizard into a red Persistent lizard for lower and lower initial percentage of adversaries and for the following training regimes.

1. When training the adversaries starting from a random initialization of the weights and set $\lambda_2 = 0.01$
2. When training the adversaries starting from the same weights of the original cells, $w_{new} = w_{old}$, and set $\lambda_2 = 0.01$
3. When training the adversaries starting from the same weights of the original cells, $w_{new} = w_{old}$, and set $\lambda_2 = 0$
4. When training the adversaries starting from a random initialization of the weights and set $\lambda_2 = 0$

We train each model until it has a loss < 0.01, which we found was an appropriate value to have visually indistinguishable images.

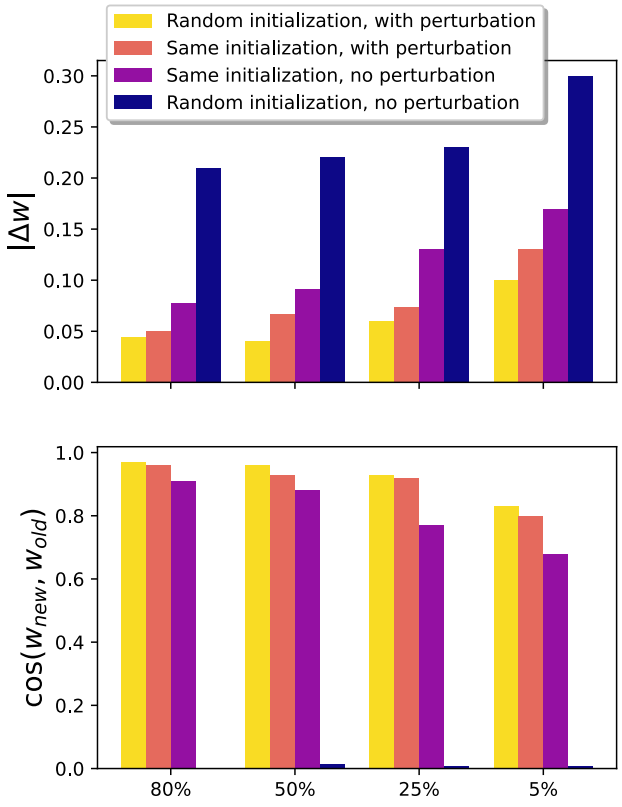

Figure 15: $|\Delta w|$ and cosine similarity measures in different training regimes. By penalizing $|\Delta w|$ we manage to find adversaries close to the original cells even when starting from a random initialization. Furthermore, as we decrease the number of initial cells, $|\Delta w|$ increases and the cosine similarity decreases.

Related Work

The paper *Adversarial Reprogramming of Neural Cellular Automata* Randazzo et al. (2021) laid the foundations for this work, while the talk given by Michael Levin at NeurIPS 2018 Levin (2018) provided biological insight and ideas for multiple experiments.

The field of Neural Cellular Automata is already vast, NCAs have been used in texture generation Niklasson et al. (2021), image classification Randazzo et al. (2020), and image segmentation Sandler et al. (2020). Furthermore, researchers managed to find the NCA that converges to a given image without training the NCA Chen and Wang (2020), in other words, the authors use a neural net to encode an image in the weights of a NCA. Other relevant works include the application of NCA to reaction-diffusion systems Mordvintsev et al. (2021), in this case the learned rule is more general because it doesn't depend on the structure of the grid, this allows a NCA trained on a 2D grid to be used on different geometries.

The kinds of adversarial attacks shown in this paper stem from the Generative Advesarial Networks Goodfellow et al. (2014) area of research and in particular *Adversarial Reprogramming of Neural Netowrks* Elsayed et al. (2018), where a target model is kept frozen while ad-hoc inputs are used to change the functional behaviour of the original model.

Additionally, computer-to-in-vivo experiments were conducted in which organisms were developed from scratch to perform specific tasks Kriegman et al. (2020). Other examples of significantly modifying anatomical outcomes without altering the genome include lines of flatworms that regenerate with two heads following alteration of bioelectric signaling Durant F (2016).

Conclusion

We demonstrated that it is possible to change global properties of a Neural Cellular Automata, by injecting very few adversaries that gradually take over the entire organism. For some tasks, where the time is a major factor, a larger injection of adversaries will be needed; for example, if we want to make a mortal organism immortal, the adversaries must be able to take over before the organism dies. Finally, we showed that the parameters of the adversaries can be chosen to be close to the original cells that they replace.

The many similarities between Neural Cellular Automata and real biological organisms, may indicate that a more refined technique could be used to bioengineer already living organisms.

Acknowledgments

We thank Alessio Ansuini for his feedback and support.

References

Berto, F. and Tagliabue, J. (2022). Cellular Automata. In Zalta, E. N., editor, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, Spring 2022 edition.

Biswas S, A. S. (2016). Neural model of gene regulatory network: a survey on supportive meta-heuristics. *Theory Biosci*. https://pubmed.ncbi.nlm.nih.gov/27048512/.

Chen, M. and Wang, Z. (2020). Image generation with neural cellular automatas.

Durant F, Lobo D, H. J. L. M. (2016). Physiological controls of large-scale patterning in planarian regeneration: a molecular and computational perspective on growth and form.

Elsayed, G. F., Goodfellow, I., and Sohl-Dickstein, J. (2018). Adversarial reprogramming of neural networks.

Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014). Generative adversarial networks.

Kriegman, S., Blackiston, D., Levin, M., and Bongard, J. (2020). A scalable pipeline for designing reconfigurable organisms. *Proceedings of the National Academy of Sciences*, 117(4):1853–1859.

Levin, M. (2018). What bodies think about: Bioelectric computation outside the nervous system. NeurIPS 2018.

Lobo D, Solano M, B. G. L. M. (2014). A linear-encoding model explains the variability of the target morphology in regeneration.

Moore, D. G., Walker, S. I., and Levin, M. (2018). Pattern regeneration in coupled networks. ALIFE 2018: The 2018 Conference on Artificial Life:204–205.

Mordvintsev, A., Randazzo, E., and Niklasson, E. (2021). Differentiable programming of reaction-diffusion patterns.

Mordvintsev, A., Randazzo, E., Niklasson, E., and Levin, M. (2020). Growing neural cellular automata. *Distill*. https://distill.pub/2020/growing-ca.

Nanos V, L. M. (2021). Multi-scale chimerism: An experimental window on the algorithms of anatomical control. *Cells Dev.* Epub ahead of print, https://pubmed.ncbi.nlm.nih.gov/34974205/.

Neumann, J. V. and Burks, A. W. (1966). *Theory of Self-Reproducing Automata*. University of Illinois Press, USA.

Niklasson, E., Mordvintsev, A., Randazzo, E., and Levin, M. (2021). Self-organising textures. *Distill*.

Randazzo, E., Mordvintsev, A., Niklasson, E., and Levin, M. (2021). Adversarial reprogramming of neural cellular automata. *Distill*. https://distill.pub/selforg/2021/adversarial.

Randazzo, E., Mordvintsev, A., Niklasson, E., Levin, M., and Greydanus, S. (2020). Self-classifying mnist digits. *Distill*. https://distill.pub/2020/selforg/mnist.

Sandler, M., Zhmoginov, A., Luo, L., Mordvintsev, A., Randazzo, E., and y Arcas, B. A. (2020). Image segmentation via cellular automata.