Critically Pre-Trained Neural Cellular Automata as Robot Controllers

Etienne Guichard

Master's Thesis

Cognitive Robotics
Delft University of Technology
Netherlands

April 2024

Acknowledgement

I would like to express my sincere gratitude to my supervisor Prof. Robert Babuska for allowing me the freedom to research such a niche field and for being ever so understanding during difficult periods. Further gratitudes are extended to Tomáš Mikolov for guiding me through the early stages of this research. To my parents and my brother, a heartfelt thank you for always being there for me, and for being understanding during turbulent times. Finally, I would like to express my gratefulness to my friends and girlfriend for the joy and company they provided to what could often be a solitary study.

Supervisors:

- Prof. Dr. Robert Babuska
- Dr. Tomáš Mikolov

Thesis Committee:

- Prof. Dr. R. Babuska
- Dr. Ing. J. Kober
- Dr. N. Tömen

Date of defence: 23rd of April 2024

Awarding Institute: Delft University of Technology

Student Number: 5611911

Critically Pre-Trained Neural Cellular Automata as Robot Controllers

Etienne Guichard
Cognitive Robotics
Delft University of Technology
2628 CD Delft, Netherlands

***Abstract*—Neural Cellular Automata (NCA) have recently been proposed as neuromorphic robot controllers. Despite their promising behavioural characteristics and small parameter counts, training NCA for control tasks has proven difficult and unstable. It is so tricky that curriculum-like multi-stage training programs must be used for simple control tasks. This thesis posits how criticality theory, an amalgam of statistical mechanics and neuroscience, presents a compelling case for pre-training NCA into a critical state. Mainly the increase in inter-cellular communication distance and maximization of available information. This thesis presents a novel NCA update function architecture loosely based on neuroscience and two novel interchangeable pre-training methods, one implicit and one explicit, based on criticality theory aimed at improving training performance. The new architecture and pre-training methods are tested on the Cart-Pole environment and trained with Double Deep Q-Learning (DDQN) and neuro-evolution. The new architecture improves the training speed and general performance of the NCA, whilst the two pre-training steps greatly stabilise the control task training when using DDQN. The explicit methods resulted in faster agent training than the implicit methods, but the pre-training step was prone to failure, whereas the implicit methods always succeeded. Neuro-evolution was more efficient than DDQN when training iterations and helped explain the dynamics that make NCA challenging to train. The architecture and pre-training steps were further tested on the LunarLander problem, a more complex control task. The neuro-evolution method succeeded in training but did not present excellent results, whilst DDQN failed outright to train.**

***Index Terms*—Neural Cellular Automaton, Criticality, Neuro-Evolution, Double Deep Q-Learning, Reinforcement Learning**

I. Introduction

Neural Cellular Automata (NCA) offer an enticing proposition. They promise computational complexity through simplicity by harnessing a nearly universal natural phenomenon, complex emergent behaviour through localised interactions. However, as is true for many nascent fields, a plethora of hurdles need to be overcome before meaningful state-of-the-art results can be achieved.

Neural Cellular Automata are a class of algorithms comprised of spatially distributed elements -called cells- such as grids or graphs, where each cell can be an arbitrary data structure [1, 2]. Cells are updated over time through local interactions governed by a neural network. Training the neural network results in a static map of weights and biases. However, the learned local rules encode for both the development from a random lattice configuration to one suitable for computations and information processing. This has some parallels with nature. A theory called the ”genomic-bottleneck” [3] postulates that there is an information gap between the size of the DNA encoding for the neural structure of the brain and the complexity of such a structure. Instead, the DNA encodes for the local interaction between neurons during development and maturity, and the structure and computation capabilities of the brain emerge from these local interactions.

From path-finding algorithms to embodied soft robots, NCA have been proposed as an alternative solution to various typical problems in robotics [4, 5, 6]. One of the most exciting approaches, and the primary inspiration for this thesis, is using NCA as self-organising end-to-end controllers for reinforcement learning agents. In [7], the NCA acts as a self-organising controller for the classic cart pole problem. It successfully controlled the cart pole for hundreds of thousands of steps while demonstrating 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. Impressive, though it may be, the approach encountered many problems that still need to be solved. Many of these issues, including but not limited to training stability and training time, can be attributed to inadequate neural architectures. This research was built on the foundational model for stimulating cell replication, morphogenesis, and specialisation [1]. This may render the neural architecture inadequate for simulating more complex brain-like behaviour necessary for control tasks, as information processing between tissue cells and brain cells is inherently different [8].

On a theoretical basis, NCA are plausible analogues to the brain. NCA are typically implemented as recurrent deep convolutional neural networks [9]. Research has shown that individual cortical brain cells act like deep convolutional neural networks [10]. However, NCA vary vastly in information propagation mechanisms and connective range compared to cortical cells. NCA, however, effectively represents every cell as a DNN with local intercommunication,

providing a topological and structural foundation to replicate biological neuro-functionality without mimicking it.

Moreover, due to the lattice configuration of the neural cellular automata acting as a critical component of its computation, it is entirely possible for NCA to be taught to modify their lattice values to trigger different modalities. This can be seen as a meta plasticity where the Neural Network is a reservoir of computations whose modality is triggered by phase transitions in its input variables. A form of meta-learning cellular automaton has already been explored [11], though it lacks the true meta-plasticity present in our brains.

Phase transitions play an essential role in the functionality of the brain and NCA alike. Strong evidence suggests that the brain operates in a *critical* state [12, 13, 14, 15]. A critical state is a state that exists in the phase transition from random, chaotic behaviour to ordered, predictable behaviour. In Neuroscience, it is a state in which computational complexity and neuron-to-neuron communication distances are maximised. Similarly, Cellular Automata, and by extension NCA, can have states on the edge of chaos. These states, known as the fourth class of CA, are considered critical and have been theorised to all be capable of universal computation [16].

This thesis extends the work of Variengien et al. [7] in several ways. It will present a neural architecture inspired by neuroscience that improves training stability, consistency, and task performance. It will compare the training performance of gradient-based and gradient-free methods to extend the theoretical understanding of NCA. Finally, it will present two novel pre-training steps based on criticality theory that improve NCA training times and performance. It will argue that this pre-training step broadly applies to any NCA. The thesis repository can be found on GitHub [1]

II. Backgroud and Previous Work

This section focuses solely on background information as it applies to NCA. Detailed background information could be given about topics relevant to control and reinforcement learning, but this would make the length of the document untenable. The reader is encouraged to look into, or at least have basic knowledge of reinforcement learning. The specific topics in reinforcement learning that this thesis addresses will be covered in their relevant methodology section but are not the focus of the research.

A. NCA

Neural Cellular Automata are a subclass of Cellular Automata that extends their capabilities through differentiable programming. Cellular Automata can be generally defined as a four tuple (Z^d, S, N, f) where:

- $\boldsymbol{Z^d}$ is a finite or infinite lattice with d dimensions
- $\boldsymbol{S}$ is the finite set of cell states
- $\boldsymbol{N}$ is the neighborhood
- $\boldsymbol{f}$ is the update function

Each cell c_x (where $x \in X$ represents the location of the cell on the lattice) is sequentially updated at each time step t by a function f that takes in as inputs the cell in question c_x and its neighbourhood N such that:

$$c_x^{t+1} = f(c_x^t, N(c_x^t)) \tag{1}$$

All cells are synchronously updated. Additionally, c_x can be an arbitrary data structure [17, 18, 19].

NCA replace the hand-crafted update function with one that can be learned as a feed-forward neural network. Though to fully take advantage of the many tools available for differentiable programming, they are often implemented as fully convolutional neural networks with the fully connected layers implemented as 2D convolutions to compute the whole NCA update in parallel [9]. Real values typically replace the binary cell values, and cell updates become additive over time. Thus, the NCA cell update can be formulated as follows:

$$c_x^{t+1} = c_x^t + f(c_x^t, N(c_x^t)) \tag{2}$$

This makes NCA Recurrent Residual Convolutional Networks [1]. Unlike traditional Deep Neural Networks and their monolithic design, NCA typically have a minute number of parameters and derive their complexity in computation through recurrent and local interaction.

The use of NCA in robotics has been limited and scattered thus far. Attempts have been made to use them as path-finding algorithms [4, 5], distributed embodied soft robots [6], and mechanistic controllers [7]. These methods have all found advantages to NCA, such as strong generalisation capabilities in path-finding scenarios, and have laid a foundation for further work to improve on.

B. Criticality

Dynamic criticality (criticality) is a concept with origins in statistical mechanics [20]. It studies the behaviour of physical dynamic systems, such as fluid mixtures and magnet spin orientations, at the border of a second-order phase transition. A second-order phase transition is a continuous phase transition, as opposed to a first-order phase transition, where the transition is discontinuous at the transition point.

As a concept, it has successfully explained the behaviour of many localised biological systems, such as the movement of fish schools and birds' flocks [21]. In biological terms, criticality is the transition point between chaos and order. From localised interactions alone, these groups of animals display complex global behaviour as an adaptation to environmental stimuli such as predators. In a critical state, sensitivity to said stimuli is maximised as a balance between

[1] https://github.com/etimush/Critical-Neural-Cellular-Automata

order arising from strong local interactions and disorder occurring from sensitivity is reached [22].

Most pertinent to this thesis is that the brain has been shown to exist in a critical state [12, 13, 14, 15]. In this critical state, sensitivity to perturbation in the form of neuronal inputs is maximised. This, in effect, means the maximisation of information transfer distance and computational capabilities of the brain.

As counterintuitive as it may seem that a single concept can encapsulate phenomena in statistical mechanics, biology, and neuroscience, there is a good explanation. *Universality* is an intrinsic feature of all critical systems; in essence, it means that regardless of the minutiae that determine the local interactions of dynamic systems, at large enough scales, they converge onto a single dynamical behaviour governed by the large correlation distances between individual components [23].

Further evidence that lends credence to the importance of criticality in the brain comes from a study on neuromorphic networks. The authors simulated biologically realistic neural architectures and tested their learning capabilities on a memory-encoding task. They found that, despite differences in structural properties of different networks, and despite some performing better at sub-critical or over-critical states, they all performed equally well and best in a critical state [24]. Demonstrating the importance of criticality even over neural structure.

The evidence thus put forward suggests that criticality in NCA might be a vital state to achieve for maximising their computational ability. More important than even the neural architecture selected.

C. Measuring Criticality

Since NCA are represented as lattices of values, criticality needs to be measured across a lattice. Statistical mechanics offers various tools to measure criticality. These methods come from the study of the Ising model. The Ising Model is a 2-D cellular automaton that models magnetic spin orientation in metallic lattices. Each site can exist in either the +1 or -1 state and the state of each site is dictated by the spin of its four nearest neighbours and a temperature parameter [25]. Due to universality, the exact details of the local interaction do not matter; what is essential to note is that the control parameter, temperature, affects what phase the lattice is in.

Correlation Length, Lattice Thermalisation Time, and Feature Power Law Distribution are three canonical methods for measuring the Ising model's proximity to criticality. Feature power law distribution and Lattice thermalisation have their problems. With feature power law distribution requiring prohibitively large latices to compute the feature sizes and thermalisation time approaching infinity as the lattice nears criticality [25, 26]. Correlation length offers a good balance between the two.

Moreover, correlation length more accurately measures criticality as the universality theorem prescribes. Strictly speaking, critical characteristics are homogeneous across systems as the large correlation distances cause them. Despite this, the method has flaws; correlation length can only be accurately measured on infinite-sized latices, which is an obvious limitation. This is due to the finite size effect, where distances are not large enough for long-range correlation dynamics to take effect [25]. Despite this, the method still works to approximate the critical point. In an infinite-sized lattice, as the critical point is approached, the correlation length diverges towards infinity. This divergent property of the function holds for finite-sized lattices. Thus, measuring the peak of the divergence gives a reasonable estimate. Fig. 1 illustrates how ξ diverges in the Ising model as the critical temperature is approached. Measuring correlation length (ξ) can be done as such:

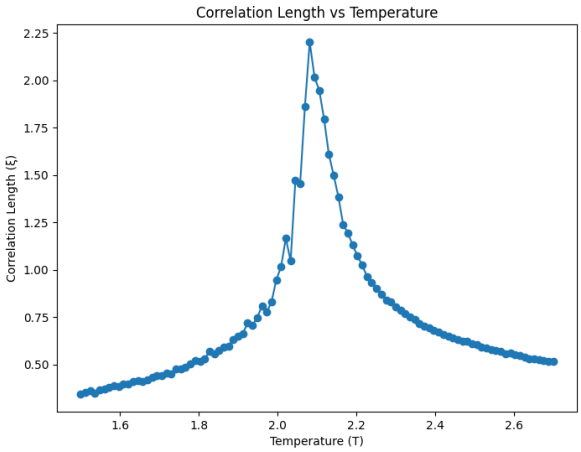


Fig. 1: Correlation length (ξ) value over a range of temperatures. Notice the strong divergence and distinct peak between temperatures 2 and 2.2; this signifies the critical temperature.

For any two sites $\{i, j\}$ the dynamic correlation function [27] C_{ij} is defined as:

$$C_{ij} = \frac{\langle (S_i(t) - \langle S_i \rangle)(S_j(t) - \langle S_j \rangle) \rangle}{\langle \sqrt{(S_i(t) - \langle S_i \rangle)^2 (S_j(t) - \langle S_j \rangle)^2} \rangle} \tag{3}$$

Where S_i and S_j are the spin values at site i, j, and $\langle\rangle$ signifies average over time.

Thus, the dynamic correlation function for the entire lattice C is defined as:

$$C = \frac{\langle \sum_{i=0}^{n} \sum_{j=0}^{n} (S_i(t) - \langle S_i \rangle)(S_j(t) - \langle S_j \rangle) \rangle}{\langle \sqrt{\sum_{i=0}^{n} (S_i(t) - \langle S_i \rangle)^2 \sum_{j=0}^{n} (S_j(t) - \langle S_j \rangle)^2} \rangle} \tag{4}$$

where n is the size of dimensions i, j.

The dynamic correlation function with respect to distance $C(r)$, $r = i - j$ is known to decay exponentially [26]:

$$C(r) \propto e^{-r/\xi} \tag{5}$$

Thus, by fitting an exponential to (4) at every temperature, with the constraint $r = i - j$, and extracting the exponent, correlation length ξ can be calculated. Fig. 2 illustrates the dynamic correlation function at every distance for temperatures $T = 2$, $T = 2.18$, $T = 3$. At $T = 2.18$, the critical temperature, the slope of the exponential decay function, is smaller, indicating a higher value of ξ.

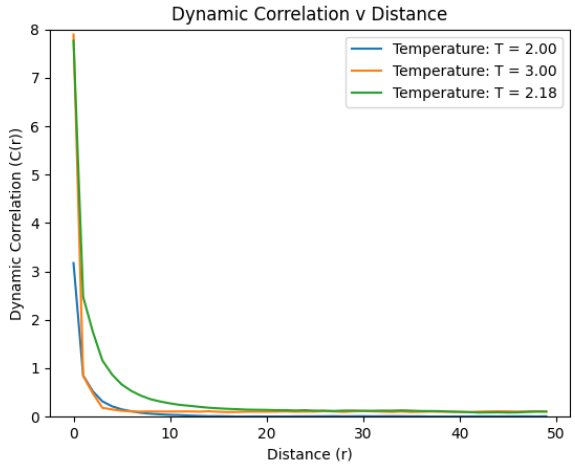
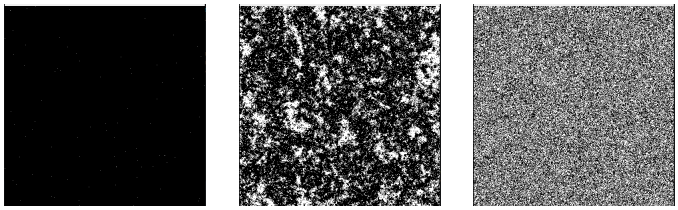


Fig. 2: Correlation function $C(r)$ with respect to r for temperatures $T = 2$, $T = 2.18$, $T = 3$. These temperatures equate to sub-critical, critical, and over-critical, respectively. The exponential decay at the critical temperature is of a lesser degree.

Fig. 3 illustrates visually what this signifies. At a sub-critical temperature (Fig. 3a), spins in the magnetic lattice align as strong local interaction dominates the dynamics, and perturbations cannot propagate. At an over-critical temperature (Fig. 3c), spins in the lattice are arranged randomly as a consequence of the over-excitability of the system. At the critical temperature (Fig. 3b), features (regions comprised of matching spins) of various sizes appear and fluctuate over time. The size of these features exhibits a power law distribution over time and at any scale due to the long correlation length between sites (which is what is meant by measuring feature power law distribution).



(a) $T = 2$ (b) $T = 2.18$ (c) $T = 3$

Fig. 3: Ising model at three different temperatures, sub-critical, critical, over-critical

To reiterate, due to the universality of criticality, this method can be extrapolated to work on real-valued latices, such as those found in NCA. The Ising model can be seen as a single-parameter Cellular Automaton, whereas NCA can be seen as an n-parameter Cellular Automaton. Thus, ξ is proposed as the loss function to be used with back-propagation to tune an NCA into a critical state.

D. Neuro-evolution

Neuro-evolution is an artificial neural network optimisation paradigm that tunes the neural architecture and its parameters through evolutionary algorithms [28]. Compared to gradient-based methods, neuro-evolution is more widely applicable. It is not necessary to learn Q-values from state action pairs as individuals can be directly evaluated on more straightforward fitness functions such as the number of steps they survived in the environment [29]. This lends it particularly well to reinforcement learning tasks where input-output pairs are hard to compute or come by. It has been shown to perform proficiently on many reinforcement learning tasks, scaling exceptionally well with CPU core counts [30].

Many approaches to neuro-volution exist; NeuroEvolution of Augmenting Topologies (NEAT) is an algorithm that evolves neural networks by adding and removing nodes and connections and tuning the weights and biases [31]. However, the NEAT algorithm typically has many tunable hyperparameters and struggles with creating large neural networks required for a more complex task. Evolutionary Deep Learning Genetic Programming (EDLGP), uses genetic programming to evolve deep neural network architectures while minimising the number of parameters needed for tuning through constructing tree ensembles of smaller neural networks [32]. Furthermore, it works with convolution layers. It can, however, struggle with creating architectures that are too large.

Unfortunately, both these approaches and any approach that modifies the topology of the neural networks used in NCA will struggle wildly. Cellular Automata generally exhibit extreme phase transitions amid small parameter changes [18]. This effect would be further magnified with topology changes incorporated in tandem. However, pure parameter search through evolutionary algorithms does offer some promise when training NCA and can help explain their dynamics

better. Furthermore, until the recent advancements in NCA, evolutionary algorithms were the primary way of optimising Cellular Automata [33, 34, 35].

Though gradients through NCA can be found and loss propagated through the parameters, their recursive nature and local interdependence can lead to complex gradient computations. Furthermore, as NCA are run for extended periods, the computation graph required to compute the gradients grows at an increasing rate. Despite the gradients being computable and the loss landscape technically being continuous, NCA erratic phase transitions lend their landscape an air of pseudo-discontinuity that can cause problems for gradient-based approaches. Examples include [7] where a pre-training step is necessary, or the NCA gets stuck at local optima, incapable of learning the cart pole environment.

Thus, neural evolution presents a solution to some of these problems. Evolutionary algorithms can deal with discontinuous fitness landscapes, handle the noise features of NCA, and avoid large computational graphs. In theory, evolutionary algorithms should be able to handle NCA end-to-end training as long as the topologies are not changed.

III. Improvements to NCA Architecture

A. General Architecture

NCA architecture has come a long way in recent years. Architecture, in this case, refers to the structure of the NN that composes the update function. The most promising advancements came from Mordvinstev et al. [1] and have served as the basis for many NCA research papers since [1, 7, 36]. The underlying theme of these papers has assembled around the replication of cell dynamics, including cell growth, pattern generation, and virus dynamics. The general architecture has predominantly been composed of three main functions: a perception step, a compute step, and an update step. The shape of the NCA (the shape of the lattice and the data each cell contains) also plays a role in its abilities. The state of the NCA refers to the dynamics the NCA exhibits due to its NN parameters and not the lattice configuration. However, since the lattice configuration affects the computations performed by the NCA, the state of the NCA can be affected by its lattice configuration.

The shape of the NCA is generally a 3-dimensional lattice composed of NxN cells, with each cell having C-real-valued channels. Thus, a C-channelled update vector must be computed and added at every step for each cell.

The perception step is primarily achieved through a convolution function and serves the purpose of propagating information across cells in a channel. It is the mechanism by which local interactions arise and is the only function in which individual cells see more information than what is contained within themselves. The kernel size dictates the communication range between cells. The kernels can be static or learnable. To illustrate, if one was interested in gradients across channels, a set of Sobel filters could be used. A perception vector is created and passed on to the compute function from these kernels and the cells' current stat. See Fig. 4.

The compute step is the central computational element of NCA. It represents the most significant number of parameters allocated to the NCA, taking in the perception vector and calculating an incremental update that is added back to each cell. It is assembled from standard differentiable building blocks such as fully connected layers (implemented as 1x1 convolutions) and non-linear activation functions. Unlike typical Neural Networks, there is no non-linear function at the final layer; this allows it to increment or decrease the cell values in the NCA as needed.

The update step acts as a form of conditioning to the outputs of the compute function. For certain NCA, behaviours such as asynchronous updates need to be forced. These behaviours are implemented in the update function as transformations to the data. In this step, the output from the compute function is also reshaped into the appropriate format to be added to the NCA cell states.

In control problems, input and output cells are selected. The environmental observations are injected into input cells that can either be affected by the NCA steps or left untouched as continuous sources of perturbations. The output cells are always affected by the NCA steps, and the control signal is read directly from these cells.

B. New Architecture

The neural architecture presented in this paper takes inspiration from [1] and [7] whilst presenting new concepts inspired by neuro-science, that aim to help improve the performance of the NCA as a controller. It is well-known that there are different types of neurons across the body that carry information from sensory organs such as the skin to the brain [37]. These neurons do more than propagate sensory information; they also condition or pre-process the signals before they reach the relevant cortex. Moreover, a variety of cortical neurons have been discovered. Though their functions aren't entirely known, their variety seems to play a crucial role in the capabilities of our minds [38].

It is not easy to fully realise the complexity and diversity of real neurons with the current paradigm in NCA. Imbuing NCA with the plethora of neuron-like structures necessary to imitate the brain comes at a tremendous computational cost compared to the current singular network shared across all cells. This is evident in the fact that multiple neural networks would have to be instantiated and trained concurrently per cell type, vastly increasing the number of parameters to be trained. With this in mind, two new functions are proposed in addition to the three standard functions of NCA: pre-process

and post-process.

The pre-process function is loosely based on the pre-processing functionality of the nervous system before information reaches the brain. Traditionally, input signals are injected raw into the input cells of the NCA before the update function steps are called. This presents the NCA with four tasks: process input, propagate the input, perform computations with the input, and produce an output signal at the output cells. The preprocessing function aims to remove the processing task from the NCA and move it to a single step. It consists of a small three-layer NN that expands the input dimensionality and then reduces it again. The resulting vector should contain additional features not present in the raw input signal. This vector is then injected into the NCA's input cell location.

The post-processing function was loosely inspired by a subgroup of cortical neurons called fast-firing neurons. While slower-firing neurons give deeper contextual information, fast-firing neurons are predominantly associated with movement and the corresponding feedback signals [39]. The readout for the control problem is still part of the NCA matrix, but the last channel where the readout is performed is calculated by a different neural network. This neural network calculates the control channel based on the values of the information channels. This separate neural network uses a convolution form of channel attention and a final 1x1 convolution. It can be seen as a form of reservoir computing, where the NCA information channels act as reservoirs of contextual information that iteratively update at every CA step. The final channel acts as a readout of the reservoir, picking the appropriate channels and combining them at the final CA step. See Fig. 4 for details.

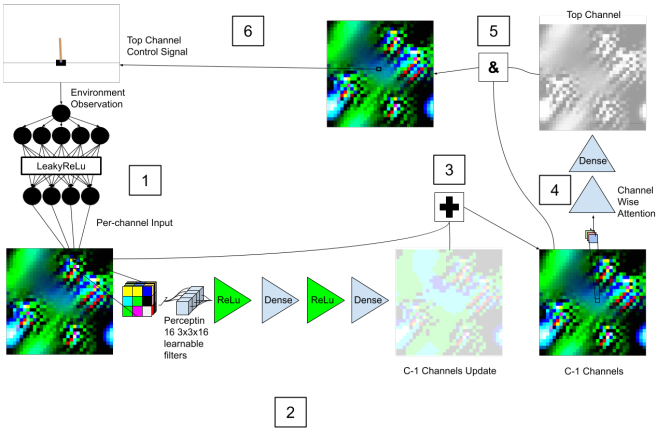


Fig. 4: New architecture for NCA update function. The NCA has C channels. **1.** The Pre-processing step transforms one observation into a C-1 (one less dimension than the number of channels C present in the NCA) dimensional vector that is injected into the NCA lattice at that observation's input cell. **2.** The NCA lattice goes through the perception and the computation steps, and a C-1 channel per pixel update is generated. **3.** The resulting update is added back to the NCA. Steps **2**-**3** are repeated N-1 times. **4.** At the Nth step, the C-1 channels of the NCA lattice are passed to the post-processing function, which generates the last channel update. **5.** The last channel update is concatenated with the C-1 channels produced in the last N-1 steps. **6.** The last channel is used as the control signal for the cart-pole environment.

IV. Training Methodology

Training directly on the control problem rarely, if ever, works. The issue arises from the interplay of stabilising the NCA dynamics and learning the correct policy. It has to transform any arbitrary lattice configuration into a stable configuration and then learn a stable configuration capable of computations. From observations, it was determined that the NCA first prioritises stabilising dynamics, followed by learning a policy. However, upon stabilising the dynamics, the model finds itself in a local optimum that is far from being capable of any control behaviour and thus never learns.

As described previously, small parametric changes lead to significant phase changes in NCA. A poignant example of this is Lenia [18], a Cellular Automata whose dynamics effectively act as a NCA. In Lenia, small changes to any parameter lead to drastically diverse behaviours. More relevant to this research, however, is that the vast majority of parameter configurations lead to an exploding or vanishing system that is stable towards an infinite horizon (exploding systems become stable as CA have a limited number of cells with maximum values; thus, when all cells reach the maximum value the system stops changing). This points to the core issue with NCA training, which is that reaching an equilibrium state is not hard. Instead, it is the default. Referring back to criticality theory, this suggests that NCA defaults to a sub-critical state with strong local interactions dominating the dynamics and very low correlation distances. The NCA training landscape appears to be peppered with immediately accessible local optima.

The necessity for a curriculum-style training regime thus becomes apparent. This should help the model by putting it in a state where varying the parameters does not cause drastic increases in the loss, enabling more effective fitness landscape exploration. Three pre-training methods will be explored: the standard one implemented in [7], and the two novel methods this thesis introduces, an implicit-criticality method and an explicit criticality method.

Training is split into two phases: pre-training and control training. For both training stages, the pre-training and control training, one training mechanism is shared. To speed up training, a pool of random lattices is initialised; a batch of lattices is then drawn from this pool and added back to it once a training step is done and the NCA has modified the lattices. One of the lattices in every batch is random to avoid the

pool of lattices becoming saturated with stable configurations, leading to the NCA forgetting how to stabilise random lattices. This allows the NCA to learn how to compute outputs on a variety of lattices and extends the number of steps the NCA can maintain a stable lattice, without incrementing the number of training steps. Unfortunately, this leads to a problem of irrecoverable lattice configurations being present in the pool, such as a lattice filled with NaNs, if training is not constrained. Thus, the loss function needs to accommodate a term for limiting the values present in the lattice.

A. Pre-control Training

1) Standard Implementation: The standard implementation is identical to [7]. The method does not appear in the paper proper, but in their code, with an explanation. The method works as such: Select a static location on the lattice for the input and output cells. At every training step, inject a random value per input cell; the loss for the backpropagation is the MSELoss between the output cell's last channel value and the average value of the input injected into the input cells. The paper theorises that this simple pre-training step teaches the NCA to transfer information towards the output cells.

Listing 1: Standard Pre-training

```
inp_cells = input_cell_positons
out_cells = output_cell_positions
nca_steps = num_nca_steps
nca = NCA(inp_cells, out_cells, nca_steps)
pool = initialise_random_grid_pool()
for _ in range(optim_steps):
    inputs = random(len(inp_cell))
    random_grid_ids = random(batch_size)
    x = pool[random_grid_ids]
    x = nca(x, inputs)
    loss = L2_loss(x[out_cells], inputs)
    pool[random_grid_ids] = x
    back_propagation(loss)
```

2) Implicit-criticality: The implicit criticality method is similar to the standard method, but there is one significant difference. At every training step, the input cell locations are selected randomly from any location on the NCA. This should make it harder for the NCA to learn the appropriate function. However, it should also induce a critical or near-critical state. The idea is that by changing the input cell location, for the NCA to reduce the loss, it needs to be able to propagate information at a distance potentially as large as the lattice. As training prolongs and the NCA start in a stable common configuration, it learns to maximise sensitivity to perturbations and propagate information long distances. This is what a critical state means.

Listing 2: Implicit Pre-training

```
out_cells = output_cell_positions
nca_steps = num_nca_steps
nca = NCA(out_cells, nca_steps)
pool = initialise_random_grid_pool()
for _ in range(optim_steps):
    inp_cell = generate_rand_inp_cells(
        num_inp_cells)
    nca.input_cells = inp_cell
    inputs = random(len(inp_cell))
    random_grid_ids = random(batch_size)
    x = pool[random_grid_ids]
    x = nca(x, inputs)
    loss = L2_loss(x[out_cells], inputs)
    pool[random_grid_ids] = x
    back_propagation(loss)
```

3) Explicit-criticality: As the name might suggest, explicit criticality pre-training aims to measure the critical state of the NCA directly. As with the Implicit-criticality function, a set of random input cell locations is selected to inject perturbation at every training step. The loss is now calculated using (4) on the entire lattice, with the dynamic correlation averaged over all information channels. The exponent, ξ, is found by fitting an exponential top the resulting data of dynamic correlation with respect to distance. In theory, the training should maximise correlation distances between lattice sites. Because no specific equation is measured, such as the average of the injected inputs, this should leave the NCA in a critical state with a larger computation reservoir where maximum information between channels is preserved.

Listing 3: Explicit Pre-training

```
out_cells = output_cell_positions
nca_steps = num_nca_steps
nca = NCA(out_cells, nca_steps=1)
pool = initialise_random_grid_pool()
grid_over_time = []
for _ in range(optim_steps):
    inp_cell = generate_rand_inp_cells(
        num_inp_cells)
    nca.input_cells = inp_cell
    inputs = random(len(inp_cell))
    random_grid_ids = random(batch_size)
    x = pool[random_grid_ids]
    for _ in range(correlation_steps):
        x = nca(x, inputs)
        grid_over_time.append(x)

    C_ij = calculate_dynamic_correlation
        (grid_over_time)
    xi = fit_exponential(C_ij)
    loss = 1/xi
    pool[random_grid_ids] = x
    back_propagation(loss)
```

B. Control Training Gradient Method

To deal with control problems, Double Deep Q-learning will be used [40]. This method aims to approximate the reward given a state and an action. The Deep Q-learning algorithm suffers from overestimation of rewards, which can lead to slower training. Double Deep Q-learning addresses this by decomposing the max operation in the target into action selection and action evaluation. The greedy policy is evaluated according to the online network, but the policy value used for gradient descent is calculated using the target network as such:

$$Q^*(s_t, a_t) \approx r_t + \gamma Q(s_{t+1}, \arg\max_{a'} Q^{'}(s_t, a_t)) \tag{6}$$

Here $Q^*(s_t, a_t)$ is the estimated state action pair value, r_t is the reward, γ is the time discount factor, s_{t+1} is the state at the next time step, and $Q^{'}(s_t, a_t)$ is target network estimated state action pair value. The target network is slowly updated over time through Polyak averaging:

$$\theta^{'} \leftarrow \tau\theta + (1-\tau)\theta^{'} \tag{7}$$

$\theta^{'}$ is the target network's parameters, θ is the policy network's parameters, and τ is the averaging rate. τ was set to 0.1 for all experiments.

A significant modification needs to be made to the training procedure. The NCA needs to learn $Q^*(s_t, a_t)$, where s_t is the state of the agent, but it also needs to learn $Q^*(s_t, a_t)$ with respect to the lattice configuration of the NCA. Given a (s_t, a_t) pair, the NCA needs to learn $Q^*(s_t, a_t)$ for a range of NCA lattice configurations s_{NCA}, this can be done by computing the average $\gamma Q(s_{t+1}, \arg\max_{a'} Q^{'}(s_t, a_t))$ over all s_{NCA}. Thus, the formulation of Double Deep Q-learning for NCA is:

$$Q^*(s_t, a_t) \approx r_t + \sum_{i=0}^{N} \frac{\gamma}{N} Q(s_{t+1}, \arg\max_{a'} Q^{'}(s_t, a_t), s^i_{NCA}) \tag{8}$$

Where N is the batch size comprising of NCA lattice configurations.

This introduces the issue of increasing the computation time by a factor of N since the agent needs to be simulated for every NCA lattice configuration. To solve this, instead of a fixed horizon for the environment, pool sampling is also used to select the starting environment state as in [7]. The basic idea is that for the exploration phase, instead of starting in an initial state and exploring for a fixed horizon or until failure, the starting state is randomly selected from a pool of previous states and run for K steps. The states reached after K steps replace the selected states in the pool. This effectively means the horizon can be infinite and comes with the benefit of adequate training for long-term stability.

Another problem with reinforcement learning methods, in general, is catastrophic forgetting [41]. Later states are added to the agent's memory as the agent improves for the Q-learning steps. The agent then effectively forgets how to operate in the initial stages as fewer initial states are sampled per Q-learning steps. A significant reduction in performance ensues. This is also true for the stabilisation of the NCA state. As the stable NCA lattices replace the random lattices in the pool, the NCA forgets how to stabilise from a random start as fewer random lattices appear in the pool sampling.

To combat this on the agent side, this thesis implements a form of early memory. The first N exploration steps are saved in an alternative memory, and every subsequent step is stored in the main memory. When sampling states for the Q-learning steps, a percentage of the states are sampled from the early memory. This should ensure that the agent does not forget about early states.

For the NCA side, the solution is much simpler. When sampling NCA lattices for the Q-learning step, a percentage of the lattices are discarded and replaced with a random matrix.

The entire training procedure can be seen in Appendix A

C. Control Training Gradient Free Method

Exploration of gradient-free alternatives to training could serve a valuable purpose for NCA. As mentioned, their propensity for phase changes owing to small parameter changes effectively makes their fitness landscape discontinuous or at least very difficult to traverse.

Evaluation is both conceptually easier and computationally cheaper. Agents can be assessed purely on their performance on the task, with no need for learning policy values. This decoupling from any conceptualisation of what the agent should learn to a pure task optimisation problem could present opportunities for NCA. Namely, opening up the solution space from the constraints of policy learning, allowing for a potentially broader subset of solutions in the fitness landscape.

Evolutionary strategies (ES) [42] will be used to train the NCA. Evolutionary strategies follow the standard Evolutionary algorithm procedure. Initiate a population of N parents, produce offspring by recombing the parents, mutate the offspring, and select the "N best offspring, or best offspring + parents, to be the next iterations parents. ES typically use natural problem-dependent representations, where the problem and search spaces are identical. For NCA, the problem space is the set of parameters that lead to the desired behaviour, and the search space is the parameters. Evolutionary strategies are adaptive, changing their mutation rate alongside their parameters in the form of coevolution.

The ES used is a (μ + λ)-ES. The new parental population is chosen from the μ previous parents and λ offspring, an elitist

strategy [42]. The mutation step is the primary differentiator between ES and EA. ES have either a mutation parameter or a set of mutation parameters; in the case used in this paper, a set of mutation parameters per layer of the Neural network is used. The mutation step starts with the selection of new mutation parameters:

$$\sigma_j^{'} = \sigma_j e^{(N(0,1)-N_j(0,1))} \tag{9}$$

Where $\sigma_j^{'}$ is the new mutation parameter for layer j, σ_j is the old mutation parameter for layer j, $N(0,1)$ is a normally distributed random variable that applies to all mutation parameters, and $N_j(0,1)$ is a newly drawn normally distributed parameter for each mutation parameter j. The mutation parameters are calculated never to go negative since a negative standard deviation is impossible. The new neural network parameters x_j in layer j are calculated as follows:

$$x_j^{'} = x_j + N(0, \sigma_j) \tag{10}$$

To mitigate the issues of drastic phase changes due to many parametric changes, an additional coevolutionary parameter, in the form of per-layer parameter mutation probability, is introduced. This parameter is self-adaptive, changed by the mutation rate, and reduces the number of mutated parameters per layer j. Additionally, when mutating, only small percentages of the neural network layers are selected for mutation. See Appendix B for pseudo code.

The crossover function used to generate the offspring is also custom. Grouping parameters into families of subsets has been shown to increase training performance effectively [43]. The custom method does not explicitly calculate these families of subsets but instead considers each layer of the neural network, with its many parameters, as a subset. Thus, point-to-point crossover is done between layers of the neural network instead of the individual parameters. See Appendix C.The entire ES procedure can be seen in Appendix D

V. Experiments

The experiments are conducted on the OpenAi Gym Cart-Pole-V2 environment. The update rules are first pre-trained using one of the aforementioned pre-training steps and further trained on the cart-pole problem. The update rules in the form of a neural network can be trained like any other neural network. Thus, a standard library, PyTorch, is used for all training. The optimiser is AdamW, a variant of Adam that resolves the low general performance of Adam-trained systems [44]. The learning rate is set to $1e-3$ with no annealing over.

A. Validating the New Architecture

One of the major problems of the previous update function architecture was its inability to train. The authors claimed a fifty per cent success rate at training. In reality, what this meant was a fifty per cent success rate in overcoming a critical local minimum that allowed the NCA to perform better than random with effectively a lower success rate at learning a decent policy.

To determine the effectiveness of the new architecture, the success rate metric will be split into two categories: training propensity and policy success rate. Training propensity will measure how likely it is that any given training step achieves a validation score of over one-hundred-and-fifty (at 150 steps, there is a 0.01% chance a random policy would reach this state), whilst policy success rate will determine how often the NCA trains towards a good policy, reaching over five hundred steps (the number of steps where the cart pole environment is considered solved).

The cart-pole environment is trained for 3500 hundred backpropagation steps. Evaluations only run for 2000 environment steps maximum, enough to account for the case where the cart pole creeps slowly to one side, reaching a failure state in over 500 environment steps. The evaluation score seen in the graphs is the number of steps the cart-pole stays upright and within bounds.

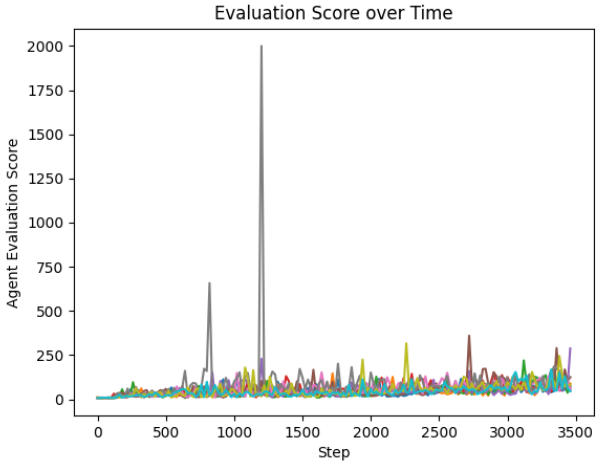
Both NCAs are trained using the same standard, static pre-training step and are further trained in the same cart pole environment. This procedure is repeated ten times per NCA.

The run-to-run performance and average performance of the standard implementation (Fig.5a, Fig.5b respectively) show a considerable struggle to train. The average slope is incremental over time, and over enough training periods, it might reach a satisfactory policy of 500 steps survived. Still, considering the relative simplicity of the cart pole environment, it performs poorly. In contrast, the new update function architecture performs significantly better. Most runs reach a good policy quickly, and some stay there. Notably, the performance in some runs degrades over time after reaching the maximum number of steps. Observations of the cart-pole in action suggest that those runs learn a policy that slowly shifts the agent to either side of the environment, where they eventually fail by going out-of-bounds. This could be a consequence of the rarity of said failure state, leading to high Q-values for actions that keep the cart pole in a steady drift.

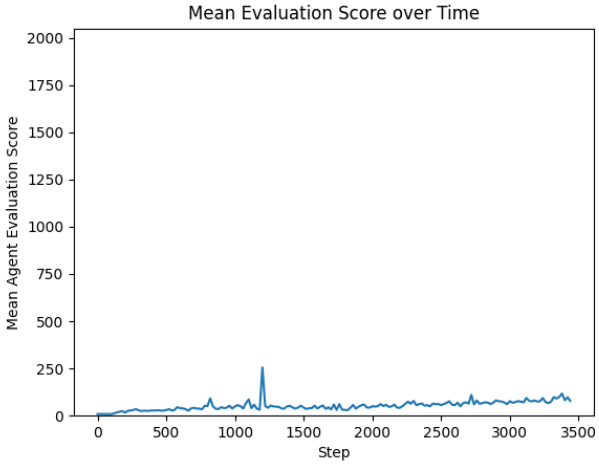
TABLE I: Evaluation Metrics Old vs New Architecture

Architecture	Training Propensity %	Policy Success Rate %	Average Score (steps)	Standard Deviation
Old	80	10	52	30
New	90	90	750	685

Table I shows that the new architecture improves slightly on the training propensity whilst improving significantly on the policy success rate. This suggests that the new model has an easier time exploring the fitness landscape, where all the training runs that managed to avoid a local minimum

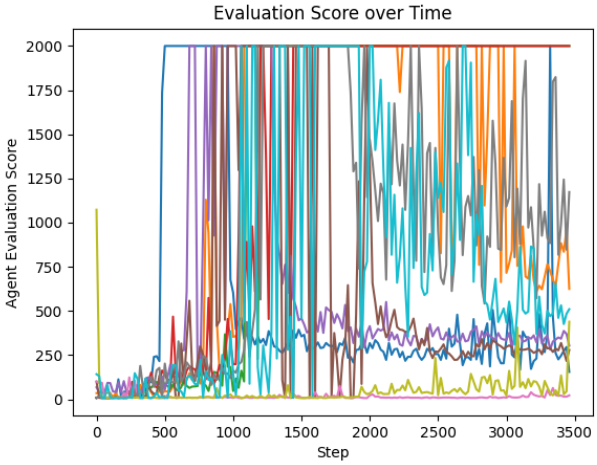


(a) Perfromance of all ten runs

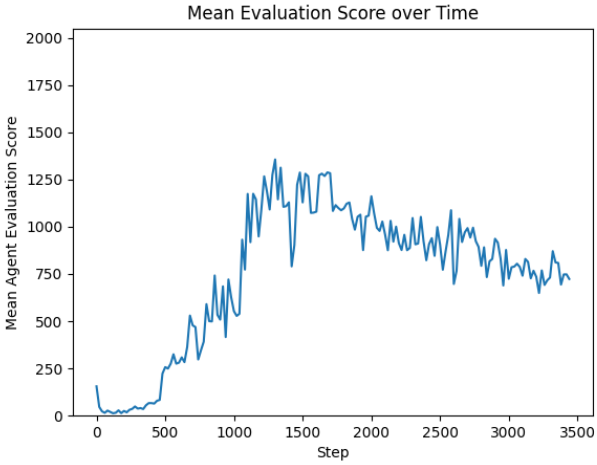


(b) Average over ten runs

Fig. 5: Performance of standard architecture over ten runs



(a) Perfromance of all ten runs



(b) Average over ten runs

Fig. 6: Performance of new architecture over ten runs

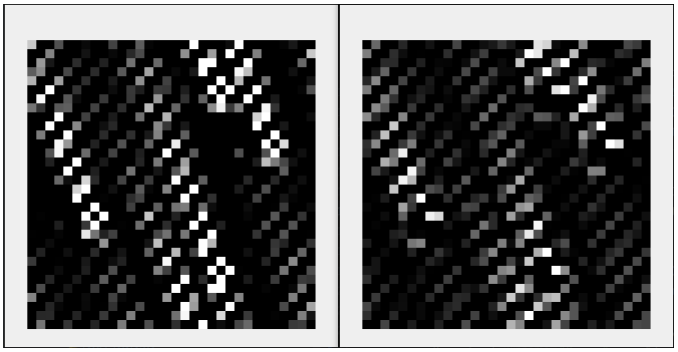
eventually perform well to some degree. The average score is roughly fifteen times better and likely much higher if the validation runs were allowed to run for longer. The high standard deviation between runs suggests that the pre-training step leads to a multitude of different "stable structures" in the NCA lattice that perform the computation. This is true for both, with the old architecture having a standard deviation of 58% of the average score, whilst, for the new architecture, it is 91%. The difference in the NCA lattice structures can be seen in Fig.7.

The top two information channels of the NCA lattices are both taken from the training of the new model, Fig.7a representing an NCA that never trained to a satisfactory level whilst Fig.7b did. The major difference in their configurations seems to be that information propagates in a diagonal versus radial manner. It seems that radial information propagation leads to better results. The reasoning behind this will be explored further when discussing the criticality-based pre-training steps.

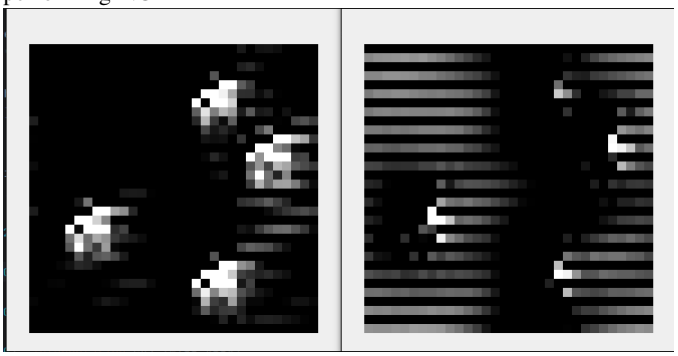
B. Ablation Studies

The ablation studies will test the different components of the new architecture. The NCA will be run without the two new components and with one of the new components at a time. The number of parameters per trial will be kept to similar levels to ensure discrepancies don't arise from larger model sizes. The old architecture is effectively the new architecture with none of the new components.

Out of both components, the post-processing step (Fig. 9) seems to have the most significant impact. Without this step, the model performs similarly to the old model. The evaluation metrics (Table II) tell a similar story, with the old



(a) Value present in the top two information channels of a poorly performing NCA

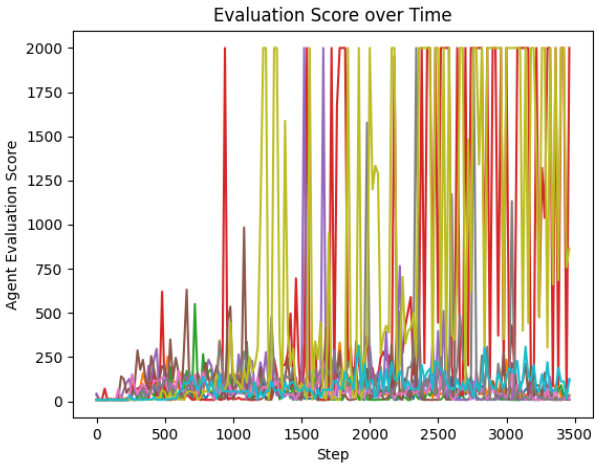


(b) Value present in the top two information channels of a well-performing NCA

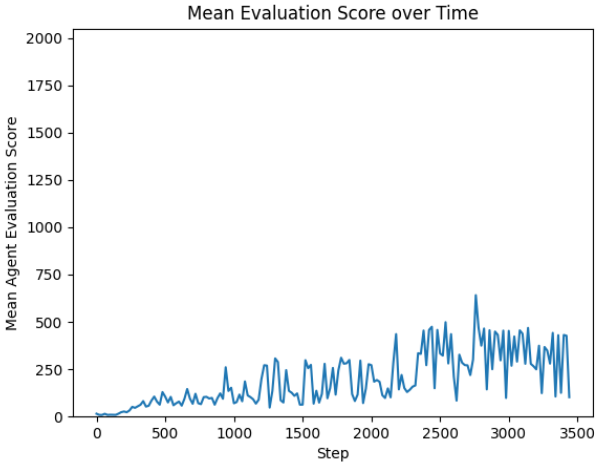
Fig. 7: Value present in the top two information channels in the lattices of a well-performing NCA and a badly-performing NCA. White pixels represent positive values, whilst black pixels present negative values, with grey in the middle.

model and the no post-processing model having identical policy success rates and similar mean scores and standard deviations. The pre-processing step has a lesser impact on its own (Figure. 8) as without it, there still is a significant improvement compared to the old model. However, it is evident that both steps together perform the best (Figure. 10) and are thus necessary for the improved performance.

Though it is hard to reason about neural networks since they are black-box models, educated guesses can be made. It is known that deeper, narrow networks perform better than shallow, wide networks [45]. By adding the extra steps to the NCA update function, the network has become deeper, whilst the post-processing step, specifically, divides the output space, taking care of the instantaneous control output separately from the rest of the network that processes information transfer. It is also possible that the input conditioning performed by the pre-processing step is insufficient to have an effect when the rest of the update function has to handle both the information propagation and the control output but supplement the structure well when only information transfer has to be performed.



(a) Scores for all ten runs of the NCA architecture without the pre-processing step, the post-processing step is still present.



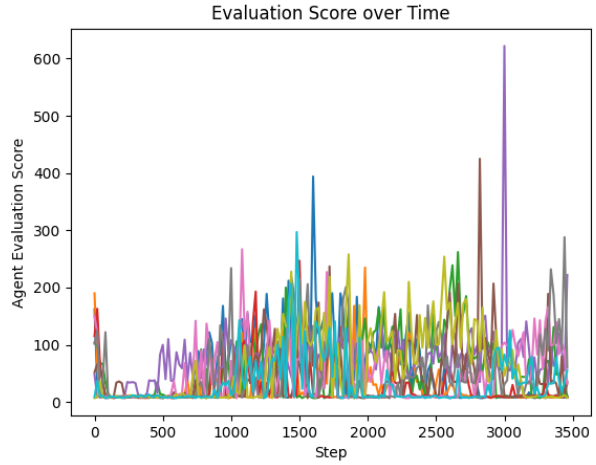
(b) Mean score for all ten runs of the NCA architecture without the pre-processing step, the post-processing step is still present.

Fig. 8: Performance of new architecture without the pre-processing step over ten runs, the post-processing step is still present.

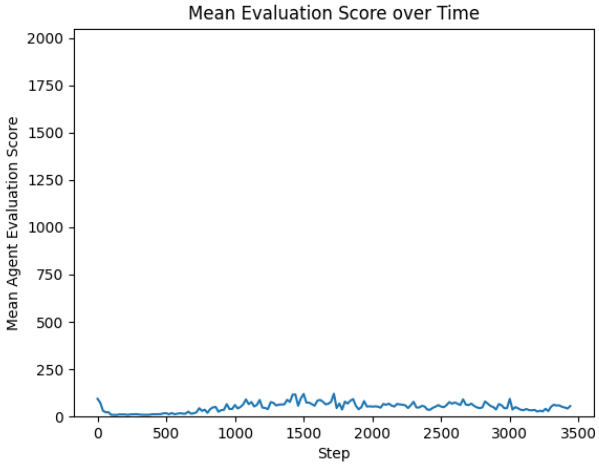
C. *Pre-traingig Implicit Criticality*

The effectiveness of the pre-training stages will be determined in the same manner as the effectiveness of the new update function architecture. For all experiments, the new architecture will be used. As before, each model will be pre-trained and trained on the control task ten times. Refer to section V-A for more details.

Fig. 11 shows the score for the ten training runs (Fig. 11a) and the mean score for those runs (Fig. 11b). The first significant difference comes in the consistency of the runs,



(a) Scores for all ten runs of the NCA architecture without the post-processing step, the pre-processing step is still present.



(b) Mean score for all ten runs of the NCA architecture without the post-processing step, the pre-processing step is still present.

Fig. 9: Performance of new architecture without the post-processing step over ten runs the pre-processing step is still present.

with all ten runs reaching the maximum evaluation steps within 1800 training steps and never losing performance after said point. There is a considerably lower variance between runs, supported by the standard deviation in Table III. One plausible explanation for this is the implicit criticality pre-training step always leads to the same behavioural characteristics of the NCA. That is to say, calculating the average of the inputs at the output cells, whilst the inputs could be in arbitrary positions, requires computational capabilities that only one permutation of the NCA update function can exhibit.

This theory is further bolstered by the pre-training step phase transition periods. Fig. 13 illustrates what this means. In essence, NCA exhibit phase transitions that lead to

TABLE II: Evaluation Metrics for the four variants of the update function tested

Architecture	Training Propensity %	Policy Success Rate %	Average Score (steps)	Standard Deviation
Old	80	10	52	30
New no pre-processing	100	60	194	314
New no post-processing	100	10	50	41
New	90	90	750	685

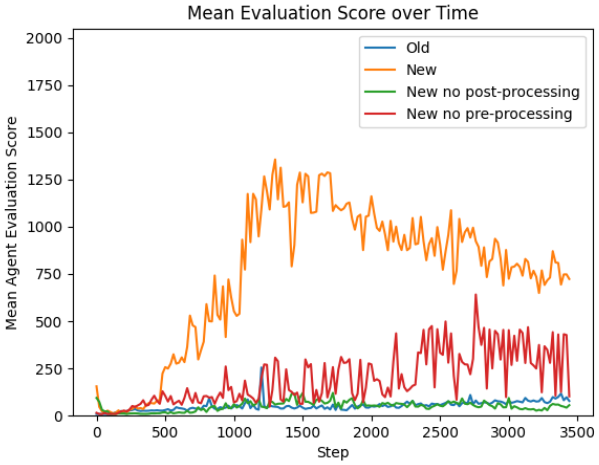
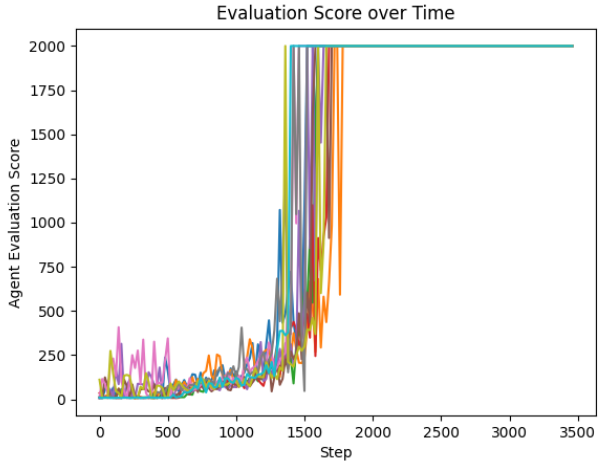


Fig. 10: Mean evaluation score over time for the old, new, new with no post-processing, and new with no pre-processing architectures for the update function.

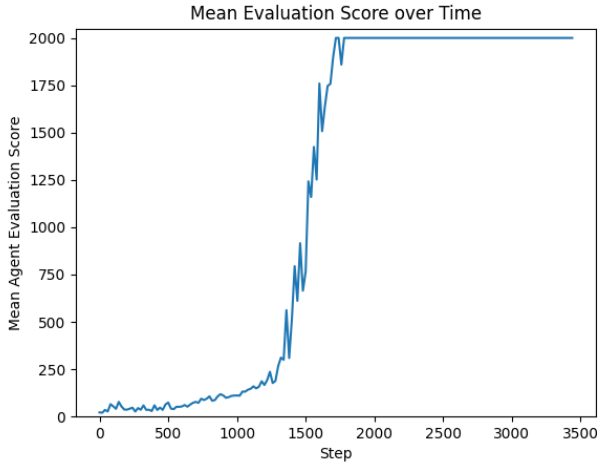
improvements in performance. These phase transitions can be seen as sudden decreases in the loss after a period of training stagnation. For both pre-training methods, the first period of rapid improvement is the NCA learning to stabilise its lattice, whilst the second period correlates with the NCA learning the task dynamics. For the standard pre-training procedure, this second phase transition period can range between 400 and 700 training steps.

In contrast, for the implicit criticality version, this second phase always happens at around 1500 training steps. The extended quiescent period between the first two phase transitions and the consistency as to when it happens suggest that there exists only one state where it can accomplish the task. This would also explain the high variance between runs of the standard pre-training method, as different solutions are found for the static input mapping; training on the control task can vary depending on how good the solution in the pre-training step was.

Comparing the correlation length (ξ) over the training



(a) Scores for all ten runs of the NCA pre-trained with the implicit criticality function.



(b) Mean score for all ten runs of the NCA pre-trained with the implicit criticality function.

Fig. 11: Performance of the new update function pre-trained on the implicit criticality method

period of both pre-training steps (Fig. 14), it can be seen that the implicit method leads to a greater correlation length than the standard methods. This strongly suggests that the implicit pre-training step is inducing a state of near criticality in the NCA.

The erratic behaviour is caused by dissonance between the actual loss function and criticality measurement. Criticality implies a maximal transfer of information, whilst the loss function merely looks at the average of the inputs, meaning some information is disregarded between cells as it does not contribute to the average. This also explains why, towards the tail end of the training, the correlation length decreases. Redundant information is no longer passed between cells. The high divergence of the critical

TABLE III: Evaluation Metrics for the standard and criticality-based pre-training methods

Pre-training method	Training Propensity %	Policy Success Rate %	Average Score (steps)	Standard Deviation
Standard pre-training	90	90	750	685
Implicit criticality pre-trainig	100	100	1198	42

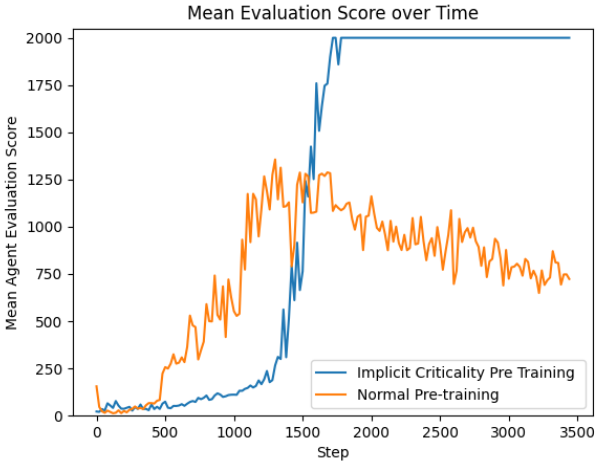


Fig. 12: Mean evaluation score over time for the standard and implicit criticality pre-training methods

points also means that small parametric changes in the NCA update function lead to high variance in the correlation length.

Comparing the top two information channels of the NCA between training runs of the standard method, where it performed well and where it performed poorly, and the top two information channels of the NCA when trained on the implicit criticality method (Fig. 15), it is clear that there are some similarities between the well-performing standard training method and the implicit criticality method. Both exhibit radial information transfer between channels, compared to the diagonal information of the poorly performing NCA. This implies that, occasionally, the standard training method achieves some form of near-criticality. The reason information would be propagated radially for a critical NCA is that criticality is rotation and transition invariant, meaning that for correlation length to be large, information needs to be propagated in all directions.

Another difference present, visible in Fig. 12, is the training pace. The NCA pre-trained on the implicit criticality method takes considerably longer to reach the same level as the standard method, about 200 steps longer. It takes another 200 backpropagation steps for it to reach its peak. The reason is hard to determine, but two theories are hereby proposed.

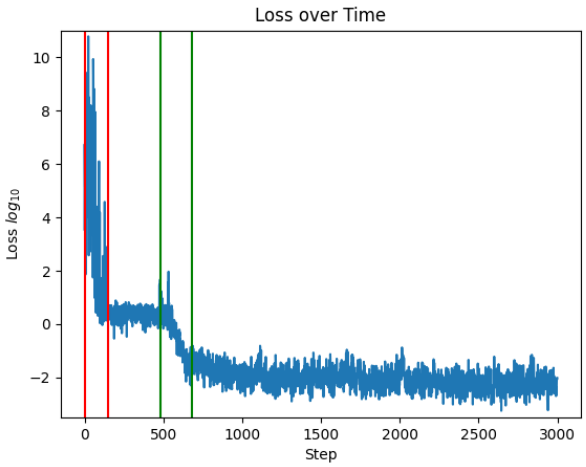


Fig. 13: Training loss over time for standard pre-training step. The period of rapid learning between the red indicates a phase change towards a stable lattice. The second phase change period between the green lines is when the NCA learns the task (mean of inputs).

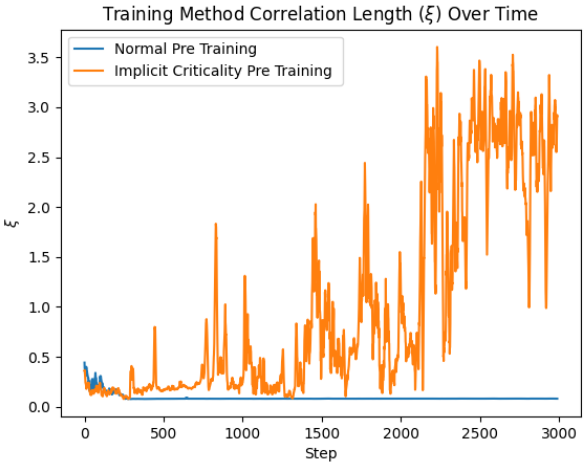
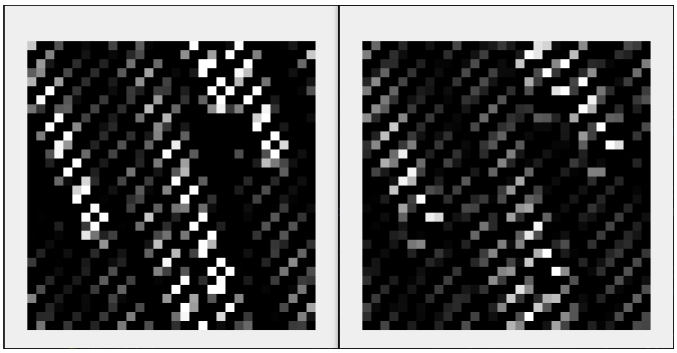


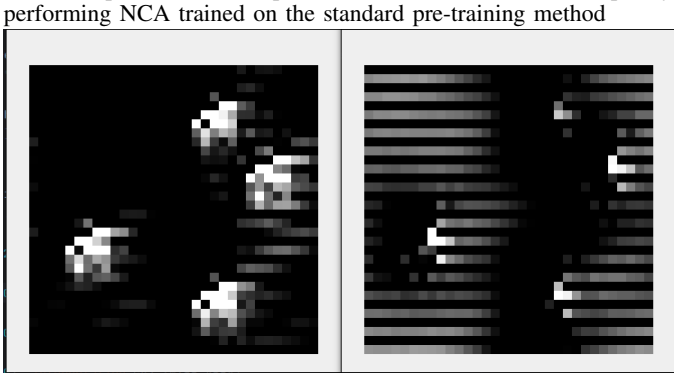
Fig. 14: Correlation length (ξ) over time for the standard and implicit criticality pre-training steps.

The more robust theory has to do with the implications of a critical state, that is, maximal information propagation. This could lead to excessive information for precisely controlling a cart pole, as not everything might be relevant. The critical state may introduce noise into the system, as it is a state in the balance between noisy chaos and local order. It is still advantageous for training as it means all information is available at the start of training, but as a consequence, training is slower.

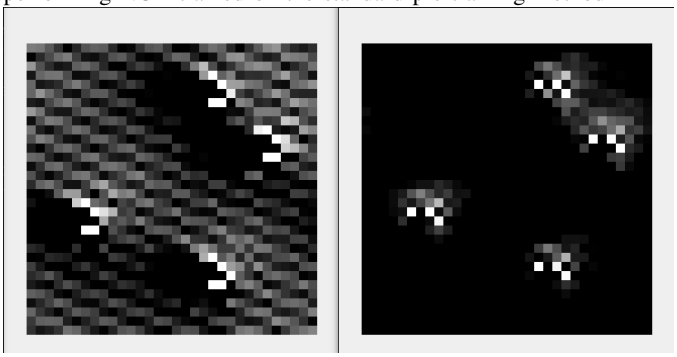
Alternatively, the slow training could be due to the static inputs of the control training process. Unlike the implicit pre-training step, the inputs for the control problem are stationary in the lattice. Thus, the NCa needs to learn the static input locations first before training for the control task. This seems unlikely, however, as one of the advantages of a critical state is the rotational and translational invariance of information transfer.



(a) Value present in the top two information channels of a poorly performing NCA trained on the standard pre-training method



(b) Value present in the top two information channels of a well-performing NCA trained on the standard pre-training method



(c) Value present in the top two information channels of an NCA trained using the implicit criticality method

Fig. 15: Value present in the top two information channels in the lattices of a well-performant NCA and a bad-performant NCA trained on the standard pre-training method, and an NCA trained on the implicit criticality pre-training method. White pixels represent positive values, whilst black pixels present negative values, with grey in the middle.

D. Pre-Training Explicit Criticality

Unlike the previous pre-training methods, the explicit method struggles with consistency. Some runs train to have high correlation distances, whilst others fail. It seems to be a complex fitness landscape to traverse and highly depends on the random start. Fig. 16 shows the correlation length over time of the explicit versus implicit criticality training method for a successful run of the explicit method. The explicit method trains considerably faster to achieve comparable values of ξ and displays considerably less variance. More importantly, however, there were no optimisation criteria apart from correlation length. Theoretically, the NCA is in a state of maximal information propagation across its cells.

Fig. 17 demonstrates the performance of an NCA trained on the explicit criticality pre-training step and compares it to the implicit step. The explicit method performs considerably better than the implicit, reaching the maximum score roughly 1000 steps earlier and significantly reducing training time. This performance increase is shown in Table IV. The explicit exhibits higher variance, although this is not detrimental.

TABLE IV: Evaluation Metrics for the explicit and implicit pre-training methods

Pre-training method	Training Propensity %	Policy Success Rate %	Average Score (steps)	Standard Deviation
Explicit pre-training	100	100	1364	57
Implicit criticality pre-trainig	100	100	1198	42

The performance increase could be attributed to a reduction in the complexity of the training process. President exists for CA acting as effective reservoirs of computation [46]. In theory, the critical state has made it so that the NCA's information channels act as a reservoir of computation. The explicit criticality method does not train the post-processing layer of the NCA. Thus, training for the cart-pole problem is reduced to the post-processing step, which is trained as a readout layer to the reservoir.

E. Neuro-Evolutionary training

The Neuro-Evolution method requires a different strategy. There is no difference between training and validation steps, as the actor's fitness can only be determined by simulating from the beginning. Since the actors are scored based on the number of steps the simulation ran, starting from a truncated state would require a large enough horizon to make it indistinguishable from running the entire simulation. If the horizon was kept small and the score accumulated from

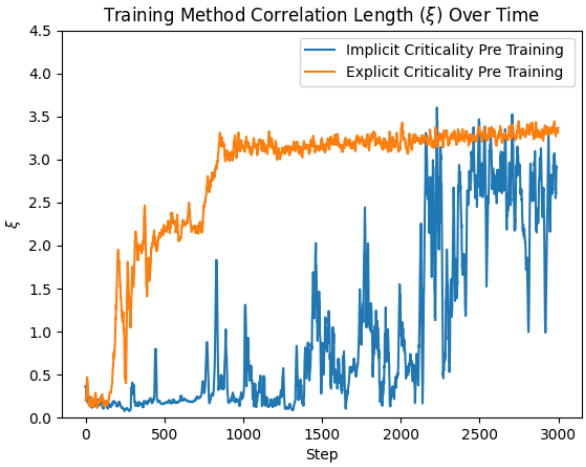


Fig. 16: Correlation length (ξ) over time for the explicit and implicit criticality pre-training steps .

the truncation point, actors randomly given a starting point further in the future would have an unrepresentative, good score.

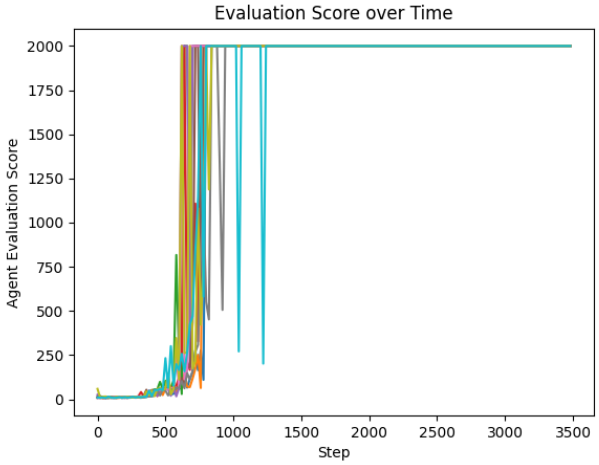
Due to differences in how backpropagation and EAs work, the EA takes considerably longer to train regarding CPU time. For this reason, the EA is only allowed to train for 50 evolution steps. This was determined experimentally to, on average, take as long as 2000 backpropagation steps.

Each of the agents' update functions is instantiated as a copy of the update function trained with the explicit criticality method. Thus, every agent in the population starts off identically.

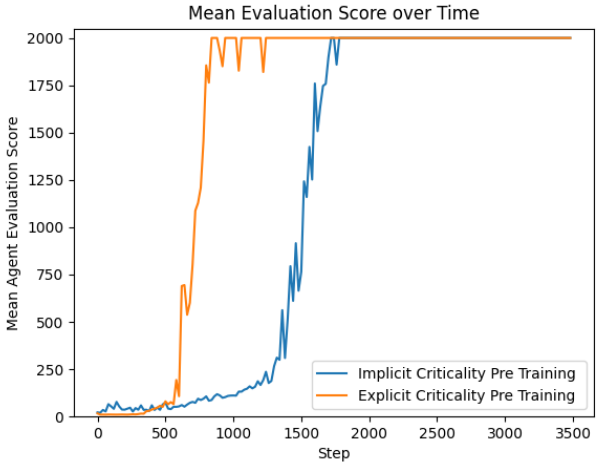
Fig. 18 shows the training performance of the evolutionary algorithm, with the update function pre-trained using the explicit criticality pre-training method. The performance, as compared to backpropagation methods, is an order of magnitude faster in terms of training steps, often reaching the 2000 environment step maximum within 30 training evolution steps.

TABLE V: Evaluation Metrics for the explicit and implicit pre-training methods

Control training method	Training Propen-sity %	Policy Success Rate %	Average Score (steps)	Standard Deviation
Best DDQN	100	100	1364	57
Neuro-evolution	100	100	1308	399

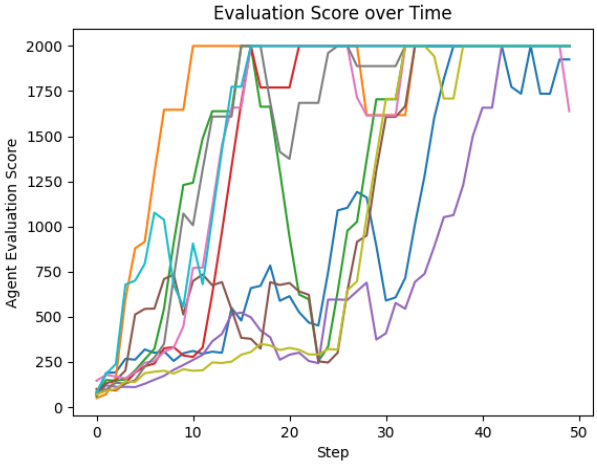


(a) Scores for all ten runs of the NCA pre-trained successfully with the explicit criticality function.

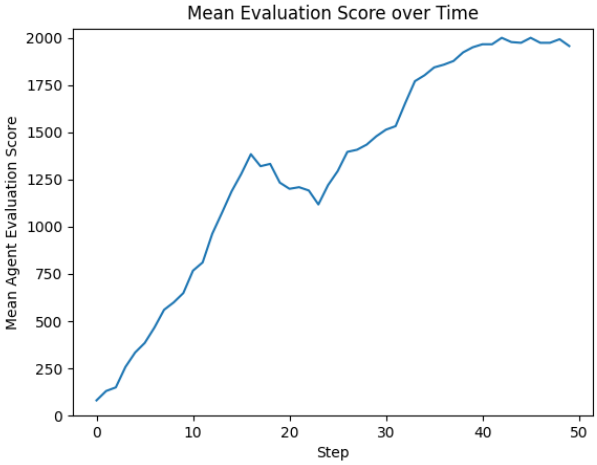


(b) Mean scores for the NCA trained on the successful explicit criticality pre-training step and the implicit pre-training step.

Fig. 17: Perfromance of the NCA trained on the explicit criticality pre-training step.



(a) Scores for all ten runs of the NCA pre-trained successfully with the explicit criticality function and trained on the control problem using neuro-evolution.



(b) Mean scores for the NCA trained on the successful explicit criticality pre-training step and trained on the control problem using neuro-evolution.

Fig. 18: Perfromance of the NCA trained on the explicit criticality pre-training step.

Table V shows the performance characteristics of the two methods. The average score is challenging to compare since the backpropagation method is trained for so long after it reaches the maximum score that it inflates it compared to the evolutionary method. However, it is evident in both Fig. 18a and Table V that the standard deviation between runs is much higher. This is most likely due to the random nature of evolutionary algorithms.

This random nature may come with the advantage of improving training performance, as it can avoid local minima by simply ”jumping” past them. The more likely explanation for the increase in training efficiency, however, is the percentage of parameters modified at each evolutionary step. At every mutation step, only 10% of layers are mutated; in every layer, only 5% of the parameters are mutated. The mutation probability may change due to self-adaptation; however, due to the rapid increase in score apparent at the beginning of training, these values may not stray far from the original 5%. This, in effect, means that, on average, 0.5% of all parameters are mutated at every evolution step. This slow mutation of parameters may offer advantages when traversing the fitness landscape by minimizing the frequency and severity of phase changes the NCA experiences as it trains. Mitigating one of the most significant issues with NCA.

F. Other Robotic Models

The experiments carried forth were extended to the OpenAI LunarLander environment. This environment is more complex than the Cart-Pole environment, requiring the agent to control a lunar lander to land in between two yellow flags on rough terrain. Environmental variables such as wind are also introduced, and the location of the flags and roughness of the terrain vary per run. This alters the policy that needs to be learned from one that solely considers the agent's state to one that also needs to consider the environment.

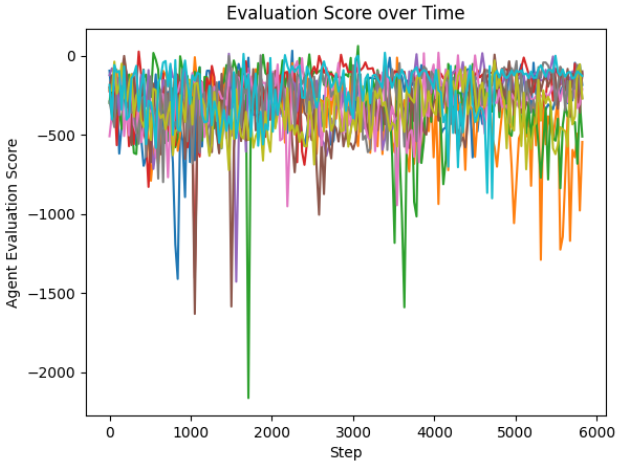
Further complexity arises from the action and observation space; there are eight observations and five possible options for control outputs, one for every thruster and one to do nothing. The NCA is now tasked with more control variables and a harder policy to learn.

The number of training steps for the neuro-evolution and backpropagation training methods was tripled to account for the increased complexity. As before, both trainings will start with a successful explicit criticality pre-training step. The network size will also remain the same, with the only noticeable difference being the input and output cell locations being changed to accommodate the new ones.

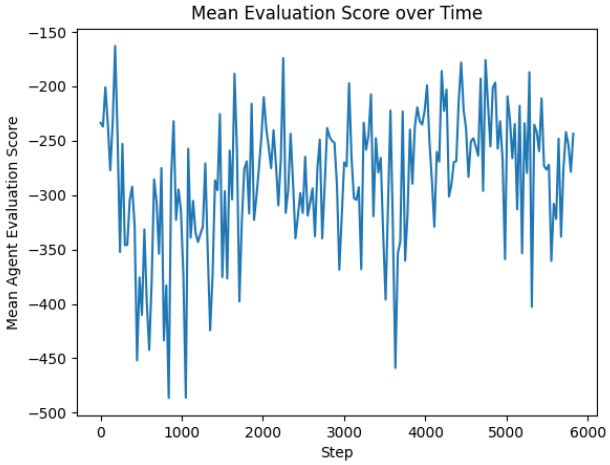
The agent is said to have learned a good policy for the lunar lander environment if it reaches a score of 200 or more for 100 runs consecutively. Thus the evaluation parameters have been changed to reflect this, with the training propensity score measured as the proportion of runs that reach a score of over 100, whilst the policy success rate is the proportion of runs that reach and maintain a score of over 200.

Fig. 19 shows the results for the NCA successfully pre-trained on the successful criticality function and trained using DDQN for the control task. The NCA fails to learn any policy that could successfully control the agent and does not show any sign of improvement over time. The increased environmental complexity leads to a considerably more challenging fitness landscape.

Fig. 20 shows the results for the NCA successfully pre-trained on the successful criticality function and trained using neuro-evolution for the control task. The NCA performs better when training using neuro-evolution, yet still struggles to find a good policy. No run manages to reach the score of 200 considered to be satisfactory. This points to a possible inherent limitation: insufficient computational complexity is achievable with the current NCA structure, be it the update function or the size of the lattice itself. Due to time and resource limitations, larger latices could not be tested. Due to the lattice configuration of the NCA playing a role in its computation, it is possible that increasing the lattice size would lead to a larger pool of possible computations that would allow for good policy to be found. This points to another core problem with the DDQN training method, primarily that it may not be capable of traversing the fitness landscape. Significant changes in parameters caused by backpropagation may be causing significant changes in policy.



(a) Scores for all ten runs of the NCA pre-trained successfully with the explicit criticality function and trained on the LunarLander problem using DDQN.



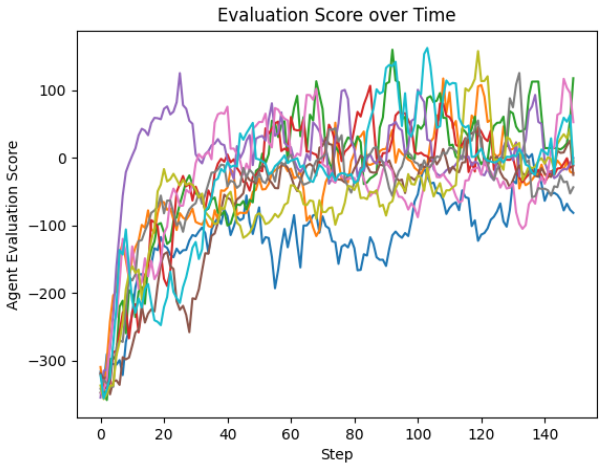
(b) Mean scores for the NCA trained on the successful explicit criticality pre-training step and trained on the LunarLander problem using DDQN.

Fig. 19: Perfromance of the NCA trained on the explicit criticality pre-training step and trained on the LunarLander problem using DDQN.

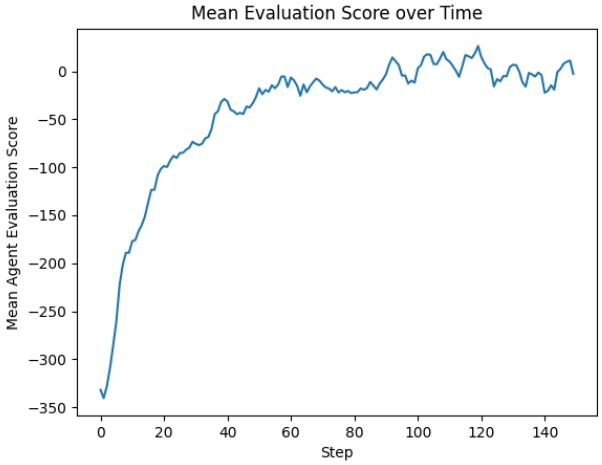
VI. Discussion

A. Contribution

The research shows that NCA architecture needs to be bespoke regarding its applications. Compared to cell division-inspired architecture, the neuro-inspired components added to the NCA architecture show a significant performance



(a) Scores for all ten runs of the NCA pre-trained successfully with the explicit criticality function and trained on the LunarLander problem using neuro-evolution.



(b) Mean scores for the NCA trained on the successful explicit criticality pre-training step and trained on the control LunarLander using neuro-evolution.

Fig. 20: Performance of the NCA trained on the explicit criticality pre-training step trained on the control LunarLander using neuro-evolution.

improvement. However, like the division-inspired architecture, training is not guaranteed unless certain conditions are met. It was also demonstrated how both architecture components were crucial for the performance benefits, with the post-processing step having the greatest effect.

Criticality is both theoretically important and valuable in practice. NCA that exhibit the defining characteristic of a critical system, mainly high correlation lengths, perform better than NCA that do not exhibit this trait. Two novel methods were proposed and shown to induce criticality in the NCA effectively. Whilst both methods worked, there was a trade-off in training reliability and training speed. The implicit criticality methods always succeeded, but training was slower than the explicit method on the control task. The explicit method was unreliable but fast at training in the control task when it succeeded.

A gradient-free method in Evolutionary Strategies was employed to test its effectiveness in training NCA. It presents a viable alternative; whilst not being faster in terms of compute time, it displays superior training performance in terms of training steps.

Finally, the NCA was trained on a more complex control problem, the OpenAI LunarLander environment, and was shown to struggle with the increased dimensionality and complexity. Whilst DDQN training was wholly ineffective, the neuro-evolution approach improved over time. Despite the improvements, a good policy was never found.

B. Further Work

Being a theoretical field of research with only recent practical applications, NCA lack the sort of advancements that lead to true leaps in functionality. Advancements such as the advent of Self-Attention in Large Language models [47]. This is not to say that attention is what NCA need but to emphasise that neural architecture has a long way to go before being genuinely applicable to real robots. Much like any neural model, theoretically, increments in network size are all that is needed to unlock capabilities. However, due to their particularities, traversing the fitness landscape with NCA becomes more challenging as the number of parameters increases. Thus, sub-modules that can help with this space traversal need to be developed. A more significant number of tests could also be conducted on existing submodules, such as self-attention or gated recurrent units, as the current neural architecture is a basic recurrent convolutional neural network.

The current real-valued approach may be inappropriate for the task of control. Our brain communicates in binary impulses, after all. Different conceptualisations, such as spiking neural networks, might be necessary to advance the field. Such models already exist but are limited [48]. Trying different models may also be necessary; justifying that our brains are the end-all and be-all of the computation does not mean that better models suited to our current technology could not be found. Models such as Hebbian networks might work well due to their exploitation of local dynamics [49]. However, it is unknown as it has not been tested.

Another major limitation is the cell homogeneity and connectivity map. There is strong evidence that cortical neurons are not homogenous, and aspects such as the myelin sheath and dendritic length affect the meta-plasticity of the brain [50]. Cortical neurons also display a form of small-world connectivity [51], as compared to the nearest neighbour model of connectivity currently used. Considerable efforts should be focused on adjusting NCA cells to display

non-homogeneous behaviour whilst maintaining the efficient training of having only one model for all the cells. It might be necessary to incorporate a small amount of global information that allows cells to act intrinsically differently.

NCA also add an additional layer of hyperparametrisation on top of the regular hyperparameters found with neural networks. The lattice dimensionality plays a role in the computational aspect, though the exact dynamics aren't known. When it comes to computations related to control, larger lattices with more considerable distances between input and output cells might afford the NCA more computational steps to perform complex tasks. The information transfer directionality might also play a role; currently, the model processes information from outer input cells towards inner output cells. It may be the case that linear or random directionality could perform better. Alternatively, isotropic models with no directional preferences [52], or hexagonal lattice models [53] could offer insight into the effects of information propagation directionality. Further tests on lattice permutations and sizes need to be conducted.

It may also be that models need to be developed that void the necessity for hyper-parametric optimization. Models that are agnostic to the size of the lattice and the directionality of the information transfer. This is unlikely, however, as it is evident that there is a positive correlation between neuron density and volume and processing capabilities.

Concerning criticality, the explicit pre-training step requires further optimisation, as it is unstable in training and prone to failure. Measuring and pre-training for criticality may be wholly irrelevant. As their namesake suggests, self-organising critical systems tend to self-organize into critical states [54]. There could be variations of NCA that have a natural propensity for functional states to be in a critical form. Regarding Lenia, recent work on adapting the system to be bound by constant energy has led to Flow-Lenia [55]. Importantly, random search through Flow-Lenia parameter space mainly yields class four Cellular Automata, which are believed to be in a critical state. Lenia and Flow-Lenia, by extension, are analogues to NCA as their update functions present alternating layers of linear and non-linear computations. Though Lenia's purpose is wholly in the pursuit of artificial life, it points to the necessity of energy conservation in NCA as a keystone feature for self-organising criticality.

C. Limitations

Though the results from this research are novel, their validity could be further solidified by conducting more runs of each experiment. As it stands, ten runs per experiment could lend itself to flukes. Especially when taking into consideration the erratic nature of NCA.

A limited attempt was made by analysing similarities in the NCA information channels to explain why certain pre-trained update functions performed better at the control task. Further insight into the dynamics of the NCA lattice could have been explored. The update function remains a black box. However, the values stored in the lattice present an opportunity to measure information flow and direction, as well as cross-channel interaction.

The results from the LunarLander experiments resulted from a direct transplantation of the model used from the cart-pole environment into the LunarLander environment. This did not account for the increased complexity of the new environment and could have been a major culprit of the poor results. More could have been done to test the abilities of the NCA by adjusting the lattice size, neural network size, or any number of other hyperparameters.

D. Ethical Considerations

As with all machine learning papers, ethical considerations and implications should be considered, even when the research is primarily theoretical. The primary ethical concern should be that of resource consumption. Currently, NCA lattice sizes are small; however, if advantages to increasing their lattice sizes are found, this could lead to an explosion in the resource consumption for training these models. The setup used for this thesis used a single, middle-range GPU. Simply doubling the lattice size would require around three GPUs as the memory consumption grows with the square of the lattice size. More complex update function architectures would also increase this computational requirement. This could lead to NCA research taking up a considerable amount of computing and thus power and resources that could, in the long run, exacerbate current environmental issues.

References

[1] A. Mordvintsev, E. Randazzo, E. Niklasson, and M. Levin, "Growing Neural Cellular Automata," *Distill*, vol. 5, no. 2, p. e23, Feb. 2020. [Online]. Available: https://distill.pub/2020/growing-ca

[2] D. Grattarola, L. Livi, and C. Alippi, "Learning Graph Cellular Automata," in *Advances in Neural Information Processing Systems*, vol. 34. Curran Associates, Inc., 2021, pp. 20 983–20 994. [Online]. Available: https://proceedings.neurips.cc/paper/2021/hash/af87f7cdcda223c41c3f3ef05a3aaeea-Abstract.html

[3] A. M. Zador, "A critique of pure learning and what artificial neural networks can learn from animal brains," *Nature Communications*, vol. 10, no. 1, p. 3770, Aug 2019. [Online]. Available: https://doi.org/10.1038/s41467-019-11786-6

[4] G. C. S. Konstantinos Ioannidis and I. Andreadis, "A path planning method based on cellular automata for cooperative robots," *Applied Artificial Intelligence*,

vol. 25, no. 8, pp. 721–745, 2011. [Online]. Available: https://doi.org/10.1080/08839514.2011.606767

[5] S. Earle, O. Yildiz, J. Togelius, and C. Hegde, "Pathfinding Neural Cellular Automata," Jan. 2023, arXiv:2301.06820 [cs]. [Online]. Available: http://arxiv.org/abs/2301.06820

[6] G. Nadizar, E. Medvet, S. Nichele, and S. Pontes-Filho, "Collective control of modular soft robots via embodied Spiking Neural Cellular Automata," Apr. 2022, arXiv:2204.02099 [cs]. [Online]. Available: http://arxiv.org/abs/2204.02099

[7] A. Variengien, S. Nichele, T. Glover, and S. Pontes-Filho, "Towards self-organized control: Using neural cellular automata to robustly control a cart-pole agent," Jul. 2021, arXiv:2106.15240 [cs]. [Online]. Available: http://arxiv.org/abs/2106.15240

[8] W. T. Fitch, "Information and the single cell," *Current Opinion in Neurobiology*, vol. 71, pp. 150–157, 2021, evolution of Brains and Computation. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0959438821001173

[9] W. Gilpin, "Cellular automata as convolutional neural networks," *Physical Review E*, vol. 100, no. 3, p. 032402, Sep. 2019. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevE.100.032402

[10] D. Beniaguev, I. Segev, and M. London, "Single cortical neurons as deep artificial neural networks," *Neuron*, vol. 109, no. 17, pp. 2727–2739.e3, Sep. 2021. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0896627321005018

[11] A. Salah and Y. Al-Salqan, "Meta-learning evolutionary artificial neural networks: by means of cellular automata," in *International Conference on Computational Intelligence for Modelling, Control and Automation and International Conference on Intelligent Agents, Web Technologies and Internet Commerce (CIMCA-IAWTIC'06)*, vol. 1, 2005, pp. 186–192.

[12] V. Zimmern, "Why brain criticality is clinically relevant: A scoping review," *Front Neural Circuits*, vol. 14, p. 54, Aug. 2020.

[13] J. M. Beggs, "The criticality hypothesis: how local cortical networks might optimize information processing," *Philos. Trans. A Math. Phys. Eng. Sci.*, vol. 366, no. 1864, pp. 329–343, Feb. 2008.

[14] J. M. Beggs and D. Plenz, "Neuronal avalanches in neocortical circuits," *J Neurosci*, vol. 23, no. 35, pp. 11 167–11 177, Dec. 2003.

[15] J. O'Byrne and K. Jerbi, "How critical is brain criticality?" *Trends Neurosci*, vol. 45, no. 11, pp. 820–837, Sep. 2022.

[16] S. Wolfram, *A New Kind of Science*. Wolfram Media, May 2002. [Online]. Available: http://www.amazon.com/exec/obidos/redirect?tag=citeulike07-20&path=ASIN/1579550088

[17] A. Adamatzky, Ed., *Game of Life Cellular Automata*. London: Springer, 2010. [Online]. Available: https://link.springer.com/10.1007/978-1-84996-217-9

[18] B. W.-C. Chan, "Lenia - Biology of Artificial Life," *Complex Systems*, vol. 28, no. 3, pp. 251–286, Oct. 2019, arXiv:1812.05433 [nlin]. [Online]. Available: http://arxiv.org/abs/1812.05433

[19] A. B. J.V. Neumann, "Theory of self-reproducing automata," *University of Illinois Press*, 1966.

[20] P. C. Hohenberg and B. I. Halperin, "Theory of dynamic critical phenomena," *Rev. Mod. Phys.*, vol. 49, pp. 435–479, Jul 1977. [Online]. Available: https://link.aps.org/doi/10.1103/RevModPhys.49.435

[21] P. Romanczuk and B. C. Daniels, *Phase Transitions and Criticality in the Collective Behavior of Animals — Self-Organization and Biological Function*. WORLD SCIENTIFIC, Dec. 2022, p. 179–208. [Online]. Available: http://dx.doi.org/10.1142/9789811260438_0004

[22] M. A. Muñoz, "Colloquium: Criticality and dynamical scaling in living systems," *Rev. Mod. Phys.*, vol. 90, p. 031001, Jul 2018. [Online]. Available: https://link.aps.org/doi/10.1103/RevModPhys.90.031001

[23] A. Franklin, "Universality explained," April 2016. [Online]. Available: https://philsci-archive.pitt.edu/12044/

[24] L. E. Suárez, B. A. Richards, G. Lajoie, and B. Misic, "Learning function from structure in neuromorphic networks," *Nature Machine Intelligence*, vol. 3, no. 9, pp. 771–786, Sep 2021. [Online]. Available: https://doi.org/10.1038/s42256-021-00376-1

[25] W. Janke, *Monte Carlo Simulations in Statistical Physics — From Basic Principles to Advanced Applications*, pp. 93–166. [Online]. Available: https://www.worldscientific.com/doi/abs/10.1142/9789814417891_0003

[26] J. Thijssen, *Classical equilibrium statistical mechanics*. Cambridge University Press, 2007, p. 169–196.

[27] D. Freedman, R. Pisani, and R. Purves, "Statistics (international student edition)," *Pisani, R. Purves, 4th edn. WW Norton & Company, New York*, 2007.

[28] F. P. Such, V. Madhavan, E. Conti, J. Lehman, K. O. Stanley, and J. Clune, "Deep neuroevolution: Genetic algorithms are a competitive alternative for training deep neural networks for reinforcement learning," *CoRR*, vol. abs/1712.06567, 2017. [Online]. Available: http://arxiv.org/abs/1712.06567

[29] P. A. Vikhar, "Evolutionary algorithms: A critical review and its future prospects," in *2016 International Conference on Global Trends in Signal Processing, Information Computing and Communication (ICGTSPICC)*, 2016, pp. 261–265.

[30] T. Salimans, J. Ho, X. Chen, S. Sidor, and I. Sutskever, "Evolution strategies as a scalable alternative to reinforcement learning," 2017.

[31] K. O. Stanley and R. Miikkulainen, "Evolving Neural Networks through Augmenting Topologies," *Evolutionary Computation*, vol. 10, no. 2, pp. 99–127, Jun. 2002. [Online]. Available: https://direct.mit.edu/evco/article/10/2/99-127/1123

[32] Y. Bi, B. Xue, and M. Zhang, "Genetic Programming-Based Evolutionary Deep Learning for Data-Efficient Image Classification," *IEEE Transactions on Evolutionary Computation*, pp. 1–1, 2022, arXiv:2209.13233 [cs]. [Online]. Available: http://arxiv.org/abs/2209.13233

[33] 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, Sep. 2018, conference Name: IEEE Transactions on Cognitive and Developmental Systems.

[34] C. Grasso and J. Bongard, "Empowered Neural Cellular Automata," in *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, Jul. 2022, pp. 108–111, arXiv:2205.06771 [cs]. [Online]. Available: http://arxiv.org/abs/2205.06771

[35] K. Horibe, K. Walker, and S. Risi, "Regenerating soft robots through neural cellular automata," *CoRR*, vol. abs/2102.02579, 2021. [Online]. Available: https://arxiv.org/abs/2102.02579

[36] E. Randazzo, A. Mordvintsev, E. Niklasson, M. Levin, and S. Greydanus, "Self-classifying MNIST Digits," *Distill*, vol. 5, no. 8, p. e00027.002, Aug. 2020. [Online]. Available: https://distill.pub/2020/selforg/mnist

[37] P. J. Bazira, "An overview of the nervous system," *Surgery (Oxford)*, vol. 39, no. 8, pp. 451–462, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0263931921001411

[38] K. Letinic, R. Zoncu, and P. Rakic, "Origin of gabaergic neurons in the human neocortex," *Nature*, vol. 417, no. 6889, p. 645 – 649, 2002, cited by: 589. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-0037030677&doi=10.1038%2fnature00779&partnerID=40&md5=f4900c4e1dd3684507252af5cf44d782

[39] N. Insel and C. A. Barnes, "Differential activation of Fast-Spiking and Regular-Firing neuron populations during movement and reward in the dorsal medial frontal cortex," *Cereb Cortex*, vol. 25, no. 9, pp. 2631–2647, Apr. 2014.

[40] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," *Proc. Conf. AAAI Artif. Intell.*, vol. 30, no. 1, Mar. 2016.

[41] R. Kemker, A. Abitino, M. McClure, and C. Kanan, "Measuring catastrophic forgetting in neural networks," *CoRR*, vol. abs/1708.02072, 2017. [Online]. Available: http://arxiv.org/abs/1708.02072

[42] H.-G. Beyer and H.-P. Schwefel, *Nat. Comput.*, vol. 1, no. 1, pp. 3–52, 2002.

[43] D. Thierens and P. A. Bosman, "Optimal mixing evolutionary algorithms," in *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation*, ser. GECCO '11. New York, NY, USA: Association for Computing Machinery, 2011, p. 617–624. [Online]. Available: https://doi-org.tudelft.idm.oclc.org/10.1145/2001576.2001661

[44] I. Loshchilov and F. Hutter, "Decoupled weight decay regularization," 2019.

[45] R. Eldan and O. Shamir, "The power of depth for feedforward neural networks," 2016.

[46] N. Babson, C. Teuscher, and Portland State University, "Reservoir Computing with Complex Cellular Automata," *Complex Systems*, vol. 28, no. 4, pp. 433–455, Dec. 2019. [Online]. Available: https://www.complex-systems.com/abstracts/v28_i04_a03/

[47] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," 2023.

[48] J. J. Farner, H. Weydahl, R. Jahren, O. H. Ramstad, S. Nichele, and K. Heiney, "Evolving spiking neuron cellular automata and networks to emulate in vitro neuronal activity," 2021.

[49] G. Amato, F. Carrara, F. Falchi, C. Gennaro, and G. Lagani, "Hebbian learning meets deep convolutional neural networks," in *Lecture Notes in Computer Science*, ser. Lecture notes in computer science. Cham: Springer International Publishing, 2019, pp. 324–334.

[50] D. Suminaite, D. A. Lyons, and M. R. Livesey, "Myelinated axon physiology and regulation of neural circuit function," *Glia*, vol. 67, no. 11, pp. 2050–2062, Nov. 2019.

[51] D. J. Watts and S. H. Strogatz, "Collective dynamics of 'small-world' networks," *Nature*, vol. 393, no. 6684, pp. 440–442, Jun. 1998.

[52] A. Mordvintsev, E. Randazzo, and C. Fouts, "Growing isotropic neural cellular automata," 2022.

[53] J. Tariq and A. Kumaravel, "Construction of cellular automata over hexagonal and triangular tessellations for path planning of multi-robots," in *2016 IEEE International Conference on Computational Intelligence and Computing Research (ICCIC)*, Dec. 2016, pp. 1–6, iSSN: 2473-943X.

[54] P. Bak, C. Tang, and K. Wiesenfeld, "Self-organized criticality," *Phys. Rev. A*, vol. 38, pp. 364–374, Jul 1988. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevA.38.364

[55] E. Plantec, G. Hamon, M. Etcheverry, P.-Y. Oudeyer, C. Moulin-Frier, and B. W.-C. Chan, "Flow-lenia: Towards open-ended evolution in cellular automata through mass conservation and parameter localization," 2023.

APPENDIX A
DOUBLE DQN PROCEDURE

Listing 4: DDQN training procedure

```
environment = make_environment(Cart-Pole-v2)
policy_nca = NCA(args)
target_nca = NCA(args)
main_memory  = initialise_memory()
early_memory =initialise_memory()
agent = DQN_AGENT(policy_nca, target_nca)
state_pool = [environment.reset() for _ in range(pool_size)]
grid_pool = [random_grid() for _ in range(pool_size)]
K = 2
for i in range(num_episodes):
    state = random_sample(state_pool,1)
    environment.set_state(state)
    for j in range (K):
        grids = random_sample(grid_pool, batch_size)
        action, new_grids = agent.select_action(state, grids)

        if random.random() < eps_threshold:
            action = random_sample(environment.action_space)

        aneal_eps()
        next_state, reward, done =  env.step(action)

        if len(early_memory) < 1000:
            early_memory.store(state, action, next_state, reward)

        else:
            main_memory.store(state, action, next_state, reward)
        update_pool(state_pool, state, next state)
        update_pool(grid_pool, grids, new_grids)
        state = next_state
        if done:
            break
    agent.optimise(early_memory, main_memory, grid_pool)
    agent.target_update()
```

Listing 5: Agent Class

```
class Agent:
    def __init__(policy_nca, target_nca):

        policy_nca = policy_nca
        target_nca = target_nca
        input_cells = initialise_input_cells()
        output_cells = initialise_output_cells()

    def select_action(state, grids):

        new_grids = policy_nca(grids, state, input_cells)
        action =  argmax(grids[0,output_cells])
        return action, new_grids
```

```
def optimise(early_memory, main_memory, grid_pool):

    transitions = smemory_sample(early_memory, main_memory)
    batch = Transition(*zip(*transitions))
    state_batch = torch.cat(batch.state)
    action_batch = torch.cat(batch.action)
    next_state_batch = torch.cat(batch.next_state)
    reward_batch = torch.cat(batch.reward)

    grids = random_sample(grid_pool, batch_size)
    state_action_values = argmax(policy_nca(grids, state_batch, input_cells)[
        output_cells])
    next_state_actions = argmax(policy_nca(grids, next_state_batch, input_cells)[
        output_cells])
    next_state_values = target_nca(grids, next_state)[output_cells].gather(
        next_state_actions)
    Q_star = next_state_values*gamma + reward_batch

    for i range(len(Q_star)):
        state_action_values[i,action_batch[i]] = Q_star[I]

    loss = 0
    for grid in grids:
        prediction = policy_nca(grid, state_batch, input_cells)
        loss += MSE_Loss(prediction.gather(action_batch), state_action_values.
            gather(action_batch)
    loss.backwards

def target_update()
    target_nca = policy_nca *tau + target_nca *(1-tau)
```

APPENDIX B
MUTATION FUNCTION

Listing 6: Mutation Function

```
def mutate(nca):
    layers = random.sample(nca.layers, len(nca.layers)/10)
    for layer in layers:
        mask = random_like(layer) < nca.mutation_probablility[layer]
        nca.mutation_rate[layer] = nca.mutation_rate[layer] * exp(stau *
            random_normal_like(nca.mutation_rate[layer]))
        nca.mutation_probability[layer] = (nca.mutation_probability[layer] + exp(stau
             * random_normal_like(nca.mutation_rate[layer]))).clamp(0,1)
        mutation = random_normal_like(layer) * nca.mutation_rate[layer]
        layer = layer + mutation*mask
```

APPENDIX C
CROSSOVER FUNCTION

Listing 7: Cross Over Function

```
def cross_over(parent_1, parent_2):
    child = parent_1.copy()
    for layer in nca.layers:
        if random.bool:
            child.layer[layer] = parent_2.layers[layer]
    return
```

APPENDIX D
ES PROCEDURE

Listing 8: ES Procedure

```
nca = NCA(args)
agents = [nca.copy() for _ in range(num_agents)]

for i in range(num_episodes):
    evaluate(agents)
    parents = truncated_selection(agents, num_parents)
    random.shuffle(parents)
    children = []
    for j in range(len(parents) - 1, step_size = 2)
        for _ in range(2):
            parent_1 = parents[j]
            parent_2 = parents[j+1]
            child = cross_over(parent_1,parent_2)
            children.append(child)

    for child in children:
        mutate(child)

    agents = parents+children
```