Recursively Fertile Self-replicating Neural Agents

Ettore Randazzo\textsuperscript{1}, Luca Versari\textsuperscript{1} and Alexander Mordvintsev\textsuperscript{1}

\textsuperscript{1}Google Research
etr@google.com

Abstract

Self-replication is a fundamental skill present in every living system. Successful living systems must be able to produce offspring that is both capable of performing a set of required tasks and of producing offspring with these same two properties, which we define as fertility. Moreover, species generally produce offspring as a fertile variation of its parents. Despite the widespread use of deep learning and neural networks in industry and academia over the last decades, self-replication with neural networks still remains largely unexplored. In this paper we train neural networks capable of encoding specific images and of producing fertile offspring with and without variation for arbitrary lengths of a genealogy. We accomplish stable self-replication by creating contractions in the parameter space of the self-replication function, and train replication with meaningful variation to give the agents a possibility of escaping these contractions in the search for other configurations that do not diverge to chaotic behaviours.

Introduction

Self-replication appears to be a pervasive property that every living system possesses. An established theory is that evolution requires replication alongside variation and selective pressure [Berg et al., 2002]. While most living systems also self-modify constantly during the lifetime of any individual, we note that self-replication is uniquely capable of exploring a larger space of possibilities at the same time due to the capacity of each individual to generate multiple and diverse offspring. Self-replication effectively hedges the survivability of a species in different and changing environments by exploring a tree of functional variations from an ancestor.

In order to generate more and more complex life and intelligent agents, perhaps we too could use self-replication as a valuable tool for discovering diverse agents. But despite the exceptional results that self-replication enabled in biology, it remains a largely under explored field of study. Evolutionary strategies, the closest cousin of self-replication research, have for the most part the replication routine disembodied from the agents, disallowing most if not any changes in the replication routine throughout the generations. Since finding interesting self-replicators is likely to involve a very large parameter space to search on, deep learning and the use of neural networks may be a good fit, but it appears that research on self-replication with deep learning is still at its infancy.

To our knowledge, only recently was self-replication first tried to be trained and studied with neural networks. Chang and Lipson (2018) were the first to explore the concept of a Neural Network Quine, a feed-forward neural network whose goal is to output its own weights given a very simple input signal. Notably, they explored configurations where the neural network was also asked to solve an auxiliary task while maintaining the capability of self-replication. Unfortunately, these neural quines appear to become completely infertile after just one self-replication step. Specifically, we mean that the parameters of the descendants diverge significantly from the ancestors over two generations and become quickly chaotic or a trivial fixpoint, and with this their performance on any given auxiliary task degrades uncontrollably. This is a problem if our goal is to make use of self-replicators for discovering interesting offspring. It would be desirable to be able to have a self-replicator capable of reproducing itself arbitrarily well, before asking it the more complex task of self-replication with variation. Gabor et al. (2019) also observed that every simple neural self-replicator, trained only to regenerate its own weights by passing the weights themselves as inputs, either diverges to infinity or to a full-zero fixpoint. Their work only explored pure self-replicators which had no other tasks, and solved the infertility problem by effectively retraining each child after each generation. We see this as a fine-tuning step where small errors in the replication step can be fixed. Fine-tuning an agent after-birth is not a problem per se, as we can see that most living creatures constantly self-modify themselves over their lifetime. However, we believe that this specific instance of fine-tuning is necessary due to the complexity of performing perfect copies in neural networks, and their inherent lack of fixed structures that could naturally behave as error-correcting mechanisms.

In this paper, we generate neural self-replicators with an auxiliary task of image encoding that generate fertile offspring without the need for fine-tuning and with the ability...
to generate variation in their offspring. To our knowledge, we are the first in generating fertile neural self-replicators without the need for fine-tuning, and the first to train for and show meaningful variation in the offspring of a neural self-replicator. We divide the experiments into three sections.

**Self-replication with only variable weights.** We start with a similar architecture to [Chang and Lipson (2018)] and show how, even with enhancements to the architecture and training regime, the offspring is not fertile, suggesting that a significantly different paradigm is required.

**Self-replication with some fixed weights.** We solve the divergence problem in a different manner from [Gabor et al. (2019)]. We demonstrate how allowing the neural network to perform a perfect copy of some of its weights while training to generate a contraction in the network’s parameter space (we will refer to this and define it as a sink later in this paper), self-replication becomes fertile and stable. Note that if the only evaluation metrics were the similarity to one’s parent and the quality of the encoded image, a solution would be to perform a perfect copy on all weights. We discourage this trivial solution by applying a loss to minimize the percentage of fixed weights and show how we find non-trivial solutions to the problem with as few as 28% fixed weights.

**Self-replication with variation.** We enhance the self-replication task to also generate fertile offspring that is different from the parent. We demonstrate how most trained configurations would self-replicate to a perfect copy of themselves, if given the same replication signals. Finally, we show how novel signal sequences give birth to novel and fertile children, some of them centered in a contraction (sink) of the self-replication function.

We conclude the paper with a discussion on the results and the current limitations of these experiments. This paper can be reproduced in the following code base: [https://tinyurl.com/58vvvest](https://tinyurl.com/58vvvest). The code base also contains extensive evaluations that are not present in this paper.

**Model**

In this section we describe the neural agents we chose to experiment with and training losses shared across experiments. Any augmentations of architectures or losses for a specific experiment will be described in the corresponding section.

At its core, the neural agent is a feed-forward neural network that accepts an image encoding input, a self-replicator encoding input and a task selection input $t$. After a function call, the agent outputs values both for the image encoding and the self-replicator output. For the image encoding task, we set $t = 1$ and set all the self-replicator inputs to zero. For the self-replication task, we set $t$ and the image encoding inputs to zero. The output is also split in two. When $t = 1$ we ignore the self-replication output, when $t = 0$ we ignore the image encoding output. See Figure 1 for a visual representation of the inputs and outputs.

![Figure 1: Graph of the inputs and outputs of the network. It can receive two kinds of inputs for two different tasks alongside a task selection value. In green, the image encoding task, in blue, the replication task.](https://tinyurl.com/58vvvest)

**Image encoding**

Given an image of size 128x128 pixels, we map each pixel position from $[0, 127]^2 \subset \mathbb{N}^2$ to the continuous range $[-1, +1]^2 \subset \mathbb{R}^2$ to form a batch of input coordinates. We now have a mapping of input coordinates $(x, y)$ to target pixel RGB values. The goal is to learn a function that given $(x, y)$ it generates the corresponding pixel value for all input-output pairs. We can then generate an entire image by passing the entire batch of input coordinates. We use the architecture described in [Sitzmann et al. (2020)] and its related initialization. This is a feed-forward neural network with sinusoidal activations in its hidden layers. We employ neither activation functions nor biases for the last layer. Throughout the paper, we use a small architecture containing 3 hidden layers of size 64 each, resulting in approximately 13k parameters.

**Self-replication**

The initial self-replication task is inspired by [Chang and Lipson (2018)]. In particular, we choose to self-replicate by passing a batch of self-replicator input signals, where each input is one-to-one mapped to a specific weight in the neural agent. We use layer-wise binary encoding, so that every weight or bias in their group (a weight is grouped by its weight matrix, a bias by its bias vector) is uniquely identified by a binary input.

**Standardization of the outputs.** Because of their initialization routines, neural networks can have significantly different magnitudes among their weight matrices and biases. We found that it is helpful to perform a post-processing standardization step for each of the weight and biases groups as follows. We generate the new group (either a weight matrix or a bias vector), compute its mean and standard deviation, and standardize them such that the final group has the same mean and standard deviation as its corresponding group on its parent. This makes it so the self-replication task learns to output, for each weight, the target displacement from the mean of its group, with standardized magnitude. This way, the complexity of the self-replication task is un-
affected by having significantly different magnitudes among groups. Output standardization is not a strict requirement, and it can be disposed of in future work.

Fixpoints and sinks
In this paper we often refer to fixpoints, $\epsilon$-fixpoints and sinks. In this context, we define a fixpoint as a configuration of a network’s parameter that, when going through a self-replication routine $f$, outputs itself. Given some network parameters $W$, $W$ is a fixpoint if and only if $f(W) = W$. As observed by Gabor et al. (2019), network weights are real-valued and it is generally impossible to have perfect precision of outputs and comparisons of floats. They extend the definition of fixpoint with the one of $\epsilon$-fixpoint, where the equality of any parameter pair is relaxed with a tolerance of $\epsilon$, that is:

$$w_i \equiv v_i \iff |w_i - v_i| < \epsilon$$

For it to be considered an $\epsilon$-fixpoint, this inequality must apply for any number of iterations with the original $W$. That is

$$\forall n > 0, W_0 \equiv f^n(W_0)$$

A sink (or contraction) is a $(\epsilon)$-fixpoint that has the additional property that there exists a neighborhood $U$ in the $(\epsilon)$-fixpoint space such that continuous applications of self-replication converge to the $(\epsilon)$-fixpoint. More formally:

$$\forall V \in U, \lim_{n \to \infty} f^n(V) \equiv W$$

Finding a sink would make self-replication stable as imperfect replication of $W$ would converge back to the original values of $W$.

Training losses
Image encoding. For the image encoding task we use an L2 norm loss on the difference between the target RGB values and the predicted ones (Figure 2). During training, we sample a minibatch of 256 random coordinates and ask the network to generate the corresponding target pixel values.

Weight divergence. A natural way to evaluate the quality of self-replication is by looking at the weight divergence between a parent and its child and to minimize this value. This can be done with any distance loss for each weight. During training, one can minimize this weight divergence (or distance) loss. Chang and Lipson (2018) employ an L2 norm as the distance loss, and Gabor et al. (2019) employ an L1 norm. However, we found this loss to be insufficient for generating fertile offspring. We observed two crucial problems: first, neural networks that use floating point precision generally cannot output the precise value for a given target weight, but at most a close approximation of it. This approximation also gets worse the more weights the network is asked to memorize. Second, this error propagates over generations and usually the grandchildren will already be chaotic and not resemblant to their ancestor. We try to solve this problem by performing a more complex training regime using a sink loss.

Sink loss. We aim to find a sink. A network exhibiting a sink would be resistant to a certain amount of noise caused by the self-replication mechanism. We can optimize for this: at each training step, instead of generating a child from the original network, we sample a random neighbor where to each weight we add Gaussian noise with a standard deviation $sd$ times the standard deviation of its group. Then, we self-replicate the neighbor and add a weight-divergence (L2) loss on this child with the original network as target (Figure 2). Note that we show that this sink loss is not sufficient for making a network with fully variable weights fertile, but suffices when some weights are fixed.

Evaluation
Whenever we show a series of images, they will be generated as in Figure 3: the network generates the image first and then self-replicates. Then we repeat with the child. We found that showing images and their qualitative degradation is generally more informative than looking at quantitative metrics such as average pixel divergence loss, for understanding the quality of the generated offspring.

We also use weight divergence to see if the network is actually performing as a sink or diverging. We use two kinds
Figure 3: Explanation of how series of image generations and replications work. We query an agent for generating an image by passing the batch of \((x, y)\) coordinates. Afterwards, we query it for generating the parameters of a child by passing the batch of binary encoding inputs. We then switch to the child and repeat the process.

Figure 4: The target image our agents are tasked to encode in their parameters.

of divergence: from the parent of the network, and from the final network we generate. Since a successful run converges to a sink, we treat the last network as the tentative sink and observe whether and when the ancestry converges to it. If both the weight divergence from the parent and from the last network converge to a very small value, we then conclude we found a sink.

Self-replication with only variable weights

Throughout the paper we will use the image in Figure 4 as a target. We chose this image from the public Flower dataset (TensorFlow, 2019) by selecting an image of interesting complexity. The image is resized to 128x128 pixels.

In this section we corroborate the instability of neural self-replicators observed by Gabor et al. (2019) by performing further experiments and showing qualitative and quantitative divergence throughout generations. We observed this infertility problem with every architecture and training regime we tried. Here, we showcase the best results we found and discuss them.

Figure 5 shows the reproductive capabilities learned by four different training regimes on the same architecture. a) is trained with a sink loss with the exact same hyperparameters that will be used in the next section when we introduce fixed weights. It fails both at creating a high quality image for the original network and at finding a sink. b) is the best result we found when training to minimize the weight divergence of the child from the parent as a primary goal. The first child has a similar image but it is infertile. c) is a variant of b where we add an image encoding loss on the child, resulting

Figure 5: Results of different training regimes for the same architecture. a) Training with a sink loss; it is the same training regime used in the next section. b) Best results when training to minimize a high loss on weight divergence. c) Best results when training to minimize both weight divergence and an added RGB loss on the child. d) Typical result when training to just minimize weight divergence without any RGB loss.
in a configuration where the child can reproduce the target image arbitrarily well. This, too, is infertile. d) shows what typically happens if we forego any image encoding loss and only train for minimizing weight divergence. Generally, this also diverges. In all these four cases, the weight divergence from the parent grows after one step and then remains very high, showing the ancestries never converge. We refer to the code base to reproduce these experiments and plot weight divergence graphs.

We observed that other architectural choices such as using ReLU activations can converge to a zero fixpoint in the parameter space. We have also experimented with other types of architectures and while the resulting networks are all infertile, the images vary significantly. Other common appearances of infertile offspring are fixed color images (i.e. with ReLUs) and high frequency noise (i.e. when not standardizing the outputs but still using sinusoidal activations).

We have also tested and augmented Chang and Lipson (2018) architecture, adding standardisation of outputs and using a larger architecture than ours, but we have observed worse results. These results are reproducible in the code base.

**Self-replication with some fixed weights**

We hypothesize these networks aren’t capable of finding a sink for a combination of two reasons: finding a (non-$\epsilon$) non-trivial fixpoint is practically impossible because of the real-valued nature of neural networks’ weights. This could be solved by finding a local contraction in the parameter space that would generate a sink for a sufficiently large neighborhood. However, these contractions are very rare (for instance Gabor et al. (2019) never found one) and they are even more unlikely to appear in a parameter manifold that also achieves sufficient quality for the image encoding task. From these assumptions, it followed that if we gave the network the possibility of morphing the parameter space as it sees fit by performing a perfect copy on some of its weights, then they could generate contractions where needed. A more informal and biological interpretation is this: networks with fully-variable weights cannot easily have a reliable error-correcting mechanism that would steer some noisy weights back to their original value. By allowing to transfer some fixed structure, we allow the network to inherit some error-correcting mechanisms.

Since performing a perfect copy on all weights would result in a quite uninteresting fixpoint that would be of no use for further experiments, we add a loss that tries to minimize the amount of fixed weights used. In this section we show that it is possible to find an interesting sink by minimizing the number of fixed weights. In the next section we will show how knowing how to do this is a building block for training models that can generate fertile offspring with variation.

**Model enhancement**

We define a differentiable transition from fully variable to fully fixed weights as follows. Let $W$ be the parameters used in the neural network (weights and biases) for either self-replication and image encoding. Let $W_{\text{var}}, W_{\text{fix}}, W_{\text{sw}}$ (read “$W$ variable, fixed, switch”, respectively) be new parameters with the same dimensionality of $W$. We initialize $W$ as follows:

$$W = W_{\text{var}} \times (1 - \sigma(W_{\text{sw}})) + W_{\text{fix}} \times \sigma(W_{\text{sw}})$$

where $\times$ is the Hadamard product and $\sigma$ is the sigmoid activation, whose range is within $(0, 1)$. During self-replication $f$, we now have:

$$W'_{\text{var}} = f(W)$$

$$W'_{\text{var}} = W'_{\text{var}} \times (1 - \sigma(W_{\text{sw}})) + W_{\text{fix}} \times \sigma(W_{\text{sw}})$$

and keep $W_{\text{fix}}, W_{\text{sw}}$ fixed. During training, they are all trainable variables, therefore the training regime can optimize for the right configuration of $W_{\text{fix}}, W_{\text{sw}}$ and the initial value for $W_{\text{var}}$.

For the sink loss, we sample a neighbor with $sd = 0.02$.

**Results**

Figure 6 shows how networks trained this way generate fertile offspring - we only show the first 20 replication steps but
we confirmed the stability for 300 steps. This specific version has a 28% mean proportion of fixed weights. Figure 7 shows the distribution of this proportion. Figure 8 shows how the weight divergence of the trained network’s ancestry (blue and orange lines) converges to a negligible value and we can conclude that the offspring converges to a very small $\epsilon$-fixpoint. Since the network at step 300 must be in the sink, by observing it takes 50 replication steps to have the weight divergence from this network stabilize (orange line), we can conclude that the original network is not in the sink, but it is in its basin of attraction. Figure 8 also shows that random neighbors and their ancestry converge to stable configurations (green line and area), albeit they appear to find a larger $\epsilon$-fixpoint. We confirmed that the images generated by neighbors ancestries are all identical for the human eye. We refer to the code base for more details and images.

The value of $sd = 0.02$ we chose for the sink training is the minimum value we observed to give the network the property to make the self-replication function $f$ a real $\epsilon$-sink. For instance, with $sd = 0.01$ the network would eventually diverge.

The outputs of the self-replication task are distributed according to a Gaussian distribution with a large enough variance to be significantly different from outputting all zeros and we found the choice of $W_{var}$ to be essential for generating the proper image. We refer to the code base for further proof of these claims and a more in-depth analysis of the properties of the resulting agent. This shows that the solution found is not trivial. However, we consider this only a building block for training self-replication with variation, since if the goal were perfect self-replication, there would be much simpler ways of accomplishing it.

Self-replication with variation

Now that we have a working configuration to generate stable sinks, we can go one step further and try to generate multiple sinks and a way for the model to travel through them. Let us extend the task by asking the original network to generate different offspring, each encoding different images. Figure 9 shows the new targets. They are channel swapping operations on the original network. From left to right respectively, we have 1) the original network; this is also the target image for the original parent. 2) a right shift with wraparound of the channels (RGB to BRG). 3) a left shift with wraparound of the channels (RGB to GBR). 4) swapping the outer channels (RGB to BGR).

We now enhance the self-replication task by adding 3 new inputs: whether to right shift, left shift or swap the outer channels. If we set them all to zero, it means to replicate without variation, if we set exactly one of these inputs to 1, we ask to replicate with the respective operation and evaluate it with the respective target image. With this enhancement we can ask the network to either self-replicate without variation and to self-replicate with one of three types of variation. In this section, a sink loss refers to applying self-replication without variation to a random neighbor.

We also update the training regime as follows: we keep both image encoding loss and sink loss on the parent and add a new step where a child is generated with one of the three new variations (chosen randomly). This child goes through its respective image encoding loss and a new sink loss, where we ask neighbors of this child to converge to this child. This means we are effectively training for the generation of 4 sinks, one for each target image.

Results

Figure 10 shows the graph of the genealogy of the resulting networks. Only the blue arrows are present during training, while the rest is behavior we have not trained for. Looking at
novel transitions, we see how the offspring has meaningful variation that also preserves fertility. Despite having used four sink losses and expected to have four respective sinks, it appears we generated only three sinks. The “outer channel swap” mutation does not generate a proper sink and it slowly converges to the “left shift” mutation sink. We also observed the same slow convergence to the “left shift” sink to happen for most of the newly found states, if we repeatedly apply identity transitions (replication without variation). Finally, we also see how there are sinks when repeatedly applying one specific kind of self-replication with variation. This is undesired and we will comment on it further in the Discussion section. We refer to the code base for a more in-depth analysis of the discovered states and transitions, evidence of the non-divergence of sinks, and the equivalence of the “left shift” sink with the sink other configurations converge to.

**Related Work**

**Self-replication literature**

While applications of self-replication with deep learning may still be in their infancy, self-replication has a long history of it being theorized and explored in many different ways. We will only focus on what we believe are the most related lines of research that inspired this paper. For an excellent overview of the entire history of self-replication, we refer the reader to [Taylor and Dorin (2020)](http://direct.mit.edu/isal/proceedings-pdf/isal/33/58/1929815/isal_a_00367.pdf).

To our knowledge, von Neumann was the first to formalize and approach artificial self-replication scientifically ([von Neumann 1951](http://direct.mit.edu/isal/proceedings-pdf/isal/33/58/1929815/isal_a_00367.pdf) [Neumann and Burks 1966]), first by defining principles for self-replication to occur, and later on by developing Cellular Automata (CAs) to explore self-replication in a controlled and restricted environment. While the more general case of self-replication was largely put aside by the artificial intelligence community, CAs saw and still see consistent explorations throughout the years since their conception. We here list a few that inspired our work. Conway’s Game of Life ([Gardner 1970](http://direct.mit.edu/isal/proceedings-pdf/isal/33/58/1929815/isal_a_00367.pdf)) is likely the most famous experimentation with CAs and inspired and awed many researchers with its results starting from simple rules. [Langton 1990](http://direct.mit.edu/isal/proceedings-pdf/isal/33/58/1929815/isal_a_00367.pdf) shows how even for CAs interesting configurations happen at the edge of chaos. This is no doubt inspired by the seminal paper by [Bak et al. 1987](http://direct.mit.edu/isal/proceedings-pdf/isal/33/58/1929815/isal_a_00367.pdf) and our explorations are extremely aligned with this view. In fact, even if we actively generate and look for sinks, we aim to find variation that is at the edge of chaos, and try to train the self-replicators to replicate in a manifold in that space. Skipping ahead to very recent developments with CAs, we saw applications to the generalization of CAs to continuous do-
mains in [Raflet 2011] and Lenia (Chan, 2019; Reinke et al., 2019; Chan, 2020). Lenia in particular also explores the discovered lifeforms in great details and creates a taxonomy for them. Finally, Neural Cellular Automata (Mordvintsev et al., 2020; Randazzo et al., 2020; Niklasson et al., 2021) show how differentiable CAs can be trained with deep learning methods to accomplish very complex tasks such as regeneration, pattern formation and image self-classification. Self-replication with CAs is a fundamental building block in the CAs environment. In general, a cell in CAs contains very simple information and the complexity of the possible results is contained in the environment’s rules. Our paper goes towards the opposite approach, where the agents are very complex and the environment is kept as simple as possible. We suspect that very interesting and diverse lifeforms will arise by changing the proportions of complexity between agents and environments.

Another example of simple organisms self-replicating in a relatively more complex environment was explored by Barricelli back in the 1950s (Barricelli, 1972), where evolution can be seen in a one-dimensional world with simple scalar (and later on vectors) agents. This is generally considered seminal work for the birth of the Artificial Life community.

One seminal work that goes further into simplifying the environment and complexifying the agents is Tierra (Ray, 1992), where the agents are a group of assembly instructions and self-replicate by executing code. Variation occurs through random mutations that are human designed, and by the self-replication routines of each agent. Interestingly, parasites that use other organisms’ code to replicate are very common in these situations.

Finally, one recent example where the environment is simpler than CAs and the agents are much more complex is Gregor and Besse (2021). In their work, each cell in a 2D grid is a unique neural network that can self-replicate and interact within their neighborhood. This is aligned with the field of open endedness (Standish, 2003), where researchers try to find the requirements for having infinitely complexifying and interesting agents while minimising the requirement for human interaction. We see our research direction to be extremely aligned with this principle, although we consider human interaction to be preferable if this would significantly speed up or enable the evolution of a system.

Machine learning
Schmidhuber (1992) and later Ha et al. (2016) showed how it is possible to train networks that generate weights for other networks. The prior work in neural self-replicators (Chang and Lipson, 2018; Gabor et al., 2019) and the initial architecture in this paper can be seen as a special case of it, where, through a proper reduction of dimensionality in the input space, we use a network to generate its own weights.

Encoding images in an agent to be then decoded by passing coordinates for each pixel is an old strategy (Sims, 1991). There, a fixed evolutionary strategy is used for generating various interesting images. Similarly, one can use neural networks and evolution for evolving neural networks that encode images in the same way (Stanley, 2007; Secretan et al., 2008). One can also train image encoding with neural networks through backpropagation. This is our method of choice and we selected, as a starting point, the architecture described in Sitzmann et al. (2020).

Discussion
In this paper we trained through backpropagation a novel neural network architecture that is able to encode a given image and to produce recursively fertile and diverse offspring. We then discovered several agents that were not present during training. These agents retain their fertility and generate diverse and interesting images that are not in the training dataset. This is a promising example on how we could bootstrap, with gradient-based optimizations, complex self-replicators with initial configurations that generate interesting and diverse offspring.

Limitations. The results show typical behaviors of deep learning. For instance, the operations we used to generate the target images for the offspring were channel swapping operations. These operations would naturally be reusable on children to generate different offspring every time. However, the model we trained overfit to converge to only one child per operation, showing that it found a simpler solution that did not involve generalising to channel swapping operations. This problem is likely solvable. The deep learning literature suggests that training for more and more complex and rich tasks makes the models generalize better. One more limitation of our results is shared with the current status of the open endedness literature, where the possible extent of images that can be generated, in particular of which the network is a sink or fertile, is limited. To truly allow for open endedness, we hypothesize that, at the very least, fixed parameters need to be modifiable (perhaps more slowly) and the architecture of the neural network needs to be variable across generations.

Final thoughts. We conclude this paper by stating our motivation for this line of work. We believe that self-replicating systems may be an excellent approach for generating increasingly complex and intelligent systems. If done right, we could reduce human intervention to a minimum and let the agents themselves decide what to try next. We know that nature managed to generate successful self-replicating systems as complex as animals, but it required billions of years and the right environment for it. Timely placed human intervention may be able to speed up this process by several orders of magnitude. This paper is a small step towards that goal, where we showed how in a closed environment and through backpropagation we can create interesting initial conditions that generate fertile self-replicating agents with an interesting amount of variation.
Acknowledgements

We thank Eyvind Niklasson, Andrey Zhmoginov and Ben Laurie for their feedback and proofreading. We thank Thomas Fischbacher for the insightful discussions.

References


