Social Neural Network Soups with Surprise Minimization

Maximilian Zorn[1], Steffen Illium[1], Thomy Phan[1], Tanja Katharina Kaiser[2],
Claudia Linnhoff-Popien[1], Thomas Gabor[1]
[1] LMU Munich, [2] TU Dresden and ScaDS.AI Dresden/Leipzig
maximilian.zorn@ifi.lmu.de

Abstract

A recent branch of research in artificial life has constructed artificial chemistry systems whose particles are dynamic neural networks. These particles can be applied to each other and show a tendency towards self-replication of their weight values. We define new interactions for said particles that allow them to recognize one another and learn predictors for each other's behavior. For instance, each particle minimizes its surprise when observing another particle's behavior. Given a special catalyst particle to exert evolutionary selection pressure on the soup of particles, these 'social' interactions are sufficient to produce emergent behavior similar to the stability pattern previously only achieved via explicit self-replication training.

Introduction

Chang and Lipson (2018) originally considered neural networks that are capable of being applied to their own weights as an input, which requires some special call structure as neural networks usually have at least one weight per input parameter. Gabor et al. (2019) constructed an artificial chemistry system based on such networks as particles where their reciprocal application leads to non-trivial emergent behavior. However, they also needed to train the neural networks with some form of self-replication as an explicit goal to achieve such behavior and stabilize the overall process.

Since the initial concept, this approach has been put to some more elaborate applications by Randazzo et al. (2021) and Illium et al. (2022). We make core use of the approach presented by Gabor et al. (2021) to include additional goals beyond self-replication for each particle. There, particles alternate by being trained for self-replication (which is still necessary to achieve stability) and being trained to fulfill an additional simple goal, which is completely independent of the soup dynamics. In this paper, we apply a similar interface but train particles to 'get to know' their peers within the soup as an additional goal.

Constructing artificial chemistries from soups offers multiple research perspectives: We may be interested in rebuilding typical artificial chemistry setups using neural networks for the sake of comparing neural networks to other data structures used to encode behavior in other artificial chemistries (Gabor et al., 2022). We may also try to discover new capabilities or architectures for neural networks that may be worthwhile to use in the countless current applications of neural networks (Randazzo et al., 2021; Illium et al., 2022). In this paper, we aim for a bit of both: We use the 'additional goal technique' to introduce (to some extent) 'social' behavior to the soup of particles. We also introduce a *catalyst* particle that is able to influence the soup behavior akin to the principles of *guided self-organization* (Prokopenko, 2013). Both ideas have a rather intuitive implementation on neural network particles *and* both ideas might be useful for (multi-agent) artificial intelligence beyond observing artificial chemistries. Note, however, that we leave these more complex multi-agent systems for future work.

In our case 'social behavior' means that particles are able to recognize one another as if they have names. Particles are then trained on tasks that relate to their peer particles. We draw inspiration from the approach of *surprise minimization*; this means that particles are trained to predict the effect of interactions with other particles, thus minimizing their 'surprise' when actually meeting them. In other words, the particles increase the confidence of their impact on the environment and of the environment on themselves. In a similar manner, we also introduce an interaction of 'general judgement' between particles, where a particle is trained to estimate another particle's success at precise self-replication. In a social context this form of 'fitness estimation' may double as a form of partner selection and may eventually form the basis for deliberate information exchange, i.e., cooperation. Effectively, we are able to show that (under certain circumstances) these social interactions can replace the previously required explicit training for self-replication.

As training for these social interactions trivially requires optimizing for a moving target, convergence or organized group behavior is not easily observed. However, since we consider it to be the nature of artificial chemistries to aim for (and possibly exploit) *emergent* behavior, we implement another technique to help the soup organize: a catalyst par-

ticle. By introducing a special particle that can influence all other particles but cannot be influenced by them, we create enough pressure on the particles' evolution to observe a form of organization. With a catalyst, we can thus observe similar stability patterns than originally reported by Gabor et al. (2019) without explicit self-replication training but arising only from social dynamics. We note that self-replication as the designated result may not be the only viable task for this setting; however, with the interpretation of replication as the simplest form of natural 'generational stability', self-replication was chosen for this work.

The main contributions of this paper thus are:

- We present and evaluate a viable approach to include social interactions between neural network particles in an artificial chemistry. We show that these interactions alone lead to stability in single particles but no observable emergent behavior.
- We show that these social interactions can alleviate the need for explicit self-replication training given that evolution pressure is provided from source other than training (in our case a specially introduced catalyst particle).

Following this introduction we provide a brief overview over additional related work (since we covered the young field of neural network artificial chemistries fully in this introduction). We then introduce the formal setting of our artificial chemistry and continue with a section containing the experiments and results. Finally, we provide an outlook on future possibilities and future work to bring them about.

Related Work

For the artificial life setting there are many fields of study that qualify as related, from emergent multi-agent interactions (Ritz et al., 2021) and self-organization in multi-agent systems (Serugendo et al., 2005) to cooperative learning of networks swarms (Van den Bergh and Engelbrecht, 2000), just to name a few. In this section, however, we focus on related applications of our core interaction: surprise minimization. Surprise minimization is, in its simplest form, the difference minimization between actual and predicted sensor input. Adaptive systems minimize surprise to maintain their states in constantly changing environments by adapting predictions or changing actions. The free-energy principle provides a mathematical formulation of this idea and has the potential for a unified brain theory for action, perception, and learning of natural systems (Friston, 2010). It has seen applications mainly in cognitive neuroscience (Holmes et al., 2021), but also in control theory (Baltieri and Buckley, 2019), reinforcement learning (Fountas et al., 2020), or social cognition (Adams et al., 2022). Other researchers in artificial intelligence took up the idea as well. Der and Martius (2012) show that controllers and world models of agents can be trained with backpropagation using the prediction error, leading to self-regulated stability in the system. For example, Berseth et al. (2020) propose an unsupervised reinforcement learning approach based on the minimization of surprise to learn behaviors for a variety of tasks. As we observe, it is essential that agents are situated in dynamic environments with disruptive forces, requiring them to take actions to avoid surprise and to maintain homeostasis. Kaiser and Hamann (2022) use simple evolutionary algorithms and a reward for high prediction accuracy (i.e., low surprise) to generate a variety of collective behaviors for robot swarms and show that self-organization can be engineered by predefining predictions. In this work, our neural networks are used both as actors (i.e., influencing the weights of other particles) and hypernetworks (cf. Ha et al., 2016) to predict future inputs, i.e., weight vectors for themselves and others.

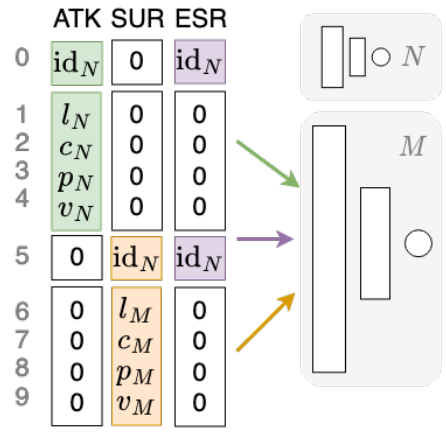

Figure 1: Schematic representation of the network input format as partially sparse (inter-)actions `ATK`, `SUR`, and `ESR`, which are formally defined as Interaction 1, Interaction 2, and Interaction 3 with either a network $\mathcal{M}$'s own weights or a different network $\mathcal{N}$'s weights.

Setting

In this work we consider particle *soups*, where one is able to observe interactions by building a population of m mutually interacting network particles $\mathcal{M}_1, \ldots, \mathcal{M}_m$. Different combinations and parameterizations of such individuals and their various (inter-)actions produce emergent behavior comparable to an artificial chemistry system (cf. Dittrich et al., 2001) or artificial life system (cf. Adami and Brown, 1994; Ofria and Brown, 1998). This means that a soup evolves over a fixed amount of epochs, i.e., evolution steps. At every epoch, different (inter-)action operators can be applied to network particles in the population with a certain chance, resulting in new particles and thus a changed soup. The general setting follows that of Gabor et al. (2021), Randazzo et al. (2021), and Gabor et al. (2022). For the sake of completeness we briefly recap the relevant material related to particle soups introduced there.

Weightwise Application

Key to finding self-replicating neural network particles is the notion of *weightwise application* (Gabor et al., 2022). It allows us to call a neural network $\mathcal{M}$ with the weights of another neural network $\mathcal{N}$ as input by simply calling $\mathcal{M}$ once for *every* weight in $\mathcal{N}$. We now assume that each neural network $\mathcal{M}$ we discuss has 10 real-numbered inputs and 1 real-numbered output so that it represents a function $\mathcal{M} : \mathbb{R}^{10} \to \mathbb{R}$. For weightwise application, we will use the first five inputs of the networks and we will discuss the use of the other inputs later in this section.

We further assume a function id which assigns to every neural network within our soup a unique identifier, e.g., $\mathrm{id}(\mathcal{M}) \in \mathbb{R}$ to represent said network $\mathcal{M}$. We treat this real-numbered values as IDs for the respective networks without them implying any structure or ordering.

We also assume that every neural network $\mathcal{M}$ in our soup adheres to the same dense-layered architecture (Goodfellow et al., 2016) and can thus be fully described by giving its weights $\overline{\mathcal{M}} \in \mathbb{R}^{26}$. Following the network's architecture, we also write the vector of weights $\overline{\mathcal{M}} = \langle v_{l,c,p} \rangle_{l,c,p}$ structured according to the layers, cells, and cell connections of the neural network so that each $v_{l,c,p} \in \mathbb{R}$ is the network weight assigned in layer l for neuron c to the connection to neuron p in the previous layer.

We can now define the application of one network to another as follows:

Definition 1 (application)**.** *Given neural networks $\mathcal{M}, \mathcal{N}$. Let $\mathcal{O} = \mathcal{M} \triangleleft \mathcal{N}$ be the application of $\mathcal{M}$ to $\mathcal{N}$ given by*

$$\overline{\mathcal{O}}_{l,c,p} = \mathcal{M}\big(\mathrm{id}(\mathcal{N}), l, c, p, \overline{\mathcal{N}}_{l,c,p}, 0, 0, 0, 0, 0\big).$$

Self-Replication

Based on the application $\triangleleft$ we can now trivially define self-application as follows:

Definition 2 (self-application)**.** *Given a neural network $\mathcal{M}$. We call the neural network $\mathcal{M}' = \mathcal{M} \triangleleft \mathcal{M}$ the self-application of $\mathcal{M}$.*

In theory, the application $\triangleleft$ has fixed points, i.e., there exist $\mathcal{M}$ so that $\mathcal{M} \triangleleft \mathcal{M} = \mathcal{M}$. In a similar way to other artificial chemistry setups (cf. Fontana and Buss, 1994), these play a special role (Gabor et al., 2019). However, since our particles are living in a continuous space, related work uses a relaxed notion called *ε-fixpoints*:

Definition 3 (ε-fixpoint, self-replication (SR))**.** *Given a neural network $\mathcal{M}$. Let $\varepsilon \in \mathbb{R}$ be the error margin for the fixpoint property.*[1] *We call $\mathcal{M}$ an ε-fixpoint iff for all l, c, p $\left|\overline{\mathcal{M}'}_{l,c,p} - \overline{\mathcal{M}}_{l,c,p}\right| < \varepsilon$ where $\mathcal{M}' = \mathcal{M} \triangleleft \mathcal{M}$. We also say that $\mathcal{M}$ is able to self-replicate.*

[1] For this paper, we assume $\varepsilon = 10^{-5}$.

Interactions

With networks interacting together in the context of a soup, Gabor et al. (2019) define the interaction of applying and substituting a randomly drawn partner particle to the predicting network as an *attack*. This interaction is given as:

Interaction 1 (attack (`ATK`))**.** *Applied to two random networks $\mathcal{M}, \mathcal{N}$ drawn from the soup at chance α, attacking substitutes the weights of the attacked network $\mathcal{M}$ with the weights given via $\mathcal{M}' = \mathcal{N} \triangleleft \mathcal{M}$.*

So far, this setting can be found almost identically in related work (cf. Gabor et al., 2022, e.g.). We now augment the setup with an interaction for minimizing surprise and an interaction for training an accurate estimation of the self-replication ability of other particles. We thus add the prediction of the effects of another particle's attack on oneself as an additional task using the other five inputs of the networks in a way comparable to the setup of Gabor et al. (2021). This allows us to train the accuracy of said prediction via standard backpropagation for supervised learning, resulting in the following new interaction:

Interaction 2 (minimize surprise (`SUR`))**.** *Applied to two random networks $\mathcal{M}, \mathcal{N}$ drawn from the soup at chance β, the particle $\mathcal{M}$ is trained to minimize the surprise S of being attacked by particle $\mathcal{N}$. To this end, we compute the predicted result $\hat{\mathcal{M}}$ of $\mathcal{N}$'s attack given via*

$$\overline{\hat{\mathcal{M}}}_{l,c,p} = \mathcal{M}\big(0, 0, 0, 0, 0, \mathrm{id}(\mathcal{N}), l, c, p, \overline{\mathcal{M}}_{l,c,p}\big)$$

and the true result $\mathcal{M}'$ of such an attack given via

$$\mathcal{M}' = \mathcal{N} \triangleleft \mathcal{M}.$$

We can then compute the surprise

$$S = \left|\left|\overline{\hat{\mathcal{M}}} - \overline{\mathcal{M}'}\right|\right|$$

for a suitable distance measurement $||\_||$ and minimize for S via stochastic gradient descent.

Finally, we introduce a second new interaction that allows particles not only to predict the results of imaginary attacks but also to predict the results of self-replication in their peer particles in order to connect our approach to the original goal of finding self-replicating particles in neural network soups.

Interaction 3 (estimate self-replicability (`ESR`))**.** *Applied to two random networks $\mathcal{M}, \mathcal{N}$ drawn from the soup at chance γ, the particle $\mathcal{M}$ is trained to minimize the prediction loss L for $\mathcal{N}$'s self-replication error. To this end, we compute the predicted self-replication error $\hat{E}$ given via*

$$\hat{E} = \mathcal{M}\big(\mathrm{id}(\mathcal{N}), 0, 0, 0, 0, \mathrm{id}(\mathcal{N}), 0, 0, 0, 0\big)$$

and the true self-replication error E given via

$$E = ||\overline{\mathcal{N}} - \overline{\mathcal{N} \triangleleft \mathcal{N}}||$$

for a suitable distance measurement $||\_||$. We can then minimize for $L = |\hat{E} - E|$ via stochastic gradient descent.

For this ESR interaction, it is important to note that its only parameter is the other network's ID given via $\mathrm{id}(\mathcal{N})$ and not any of its weights. That means that the only way a network $\mathcal{M}$ can reliably estimate another network $\mathcal{N}$'s precision at self-replication is by learning the right values 'by heart' over time. Thus, this interaction requires network particles to really incorporate knowledge about the over networks. Furthermore note that in order to keep the networks with and without ESR at the same size and architecture, we are overloading the network inputs to implement ESR. As can be seen from Interaction 3, we are passing $\mathcal{N}$'s ID to the same inputs that usually receive the other network's ID in an interaction. To clearly signal the ESR case to the network, however, we pass $\mathcal{N}$'s ID twice, i.e., through two distinct inputs to the network $\mathcal{M}$, to ensure that all internal weights that encode 'memory' of the other network's identification (in the other (inter-)actions) are being used. A schematic representation of the three different input vector formats as used for the soup actions is presented in Fig. 1.

Experiments

With the setting definitions in place, our primary concern is the construction of a stable particle soup without the direct use of self-replication (SR) training. We first consider an initial population of 10 small network particles. The layer architecture is kept reasonably simple with 10 input neurons, one hidden layer of width 2, and one output neuron for a total of 26 weights. Following the setup of Gabor et al. (2022) we do not activate the layers nor do we use a bias, as it has been shown in Illium et al. (2022) that activation functions in small weight spaces like the one considered here are negligible. For the first experiments we now only employ the actions ATK and SUR for interactions in the soup. Any weight training is computed via the Stochastic Gradient Descent (SGD) optimizer by the Python *pytorch* library (cf. Paszke et al., 2019) with a learning rate of 0.004 and a momentum parameter of 0.9.

Fig. 2 (left) shows the development of the soup after multiple epochs, although for the purpose of this demonstration we do not actually substitute the weight application predicted by the ATK actions. This prevents the system from any danger of diverging, since no real weight-destabilizing interactions are happening yet. The system would take a similar form with actual attacks of very low frequency, e.g., $\alpha \leq 0.001$, since singular ATK interactions (Interaction 1) of mostly stable particles lose impact eventually (Gabor et al., 2022). Since SUR (Interaction 2) is trained every epoch for each network with one randomly drawn partner, i.e., $\beta = 1.0$, any higher α-rate would promptly collapse the whole system, as early weight applications are imprecise enough to drastically impact the affected particles' trajectory. All following SUR interactions are therefore proportionally 'surprised' by this particle's performance, leading to a large gradient update, which propagates through all following surprise estimations. Thus, in this state without any form of explicit stability training (that is not dependent on other networks), a couple of early attacks are enough to diverge a soup collective. Without true attacks, however, we observe all particles developing steadily in place as would be expected. The remaining degree of uncertainty as indicated by the slightly curved trajectories is similar to the *goal soups* discussed by Gabor et al. (2021), where similar states of not-diverged but also not-quite-completely-converged soups were also observed. The trajectory representation of these (and the following) plots adopts the visual concept introduced by Gabor et al. (2022), where a PCA dimensionality reduction is applied to the weight vector $\overline{\mathcal{M}} \in \mathbb{R}^{26}$ to show the numerical adjustment of the weight values in a shared, easy-to-understand, two-dimensional weight space (PCA X-dimension and Y-dimension) over evolution steps (Z-dimension).

At this point we introduce a third action – estimating self-replication ability (ESR) (Interaction 3) and the minimization of the incurred loss – to the soup. Fig. 2 (right) shows the effect on the particle trajectories, as they slowly diverge from their initial positions. Similarly to how training the soup only with the SUR action without any true action always causes a degree of uncertainty for the surprise of the other networks, the act of additionally estimating the self-replication ability 'overpowers' the stability induced by the mutual SUR training and slowly propagates minor degrees of instability through the whole system. This leads to the process of divergence that is observed here (and is reported similarly by Gabor et al., 2021, 2022; Illium et al., 2022), indicated by the increasing magnitude of weight values and consequently the extreme trajectory through the weight space.

Justification for the choice of network inputs. With all relevant interactions defined and an understanding of the soup interactions in place, we briefly interject a justification of the id parameter choice in the particle's input format as shown in Fig. 1. We highlight the duplicate use of this parameter, once each for the actions ATK and SUR, and twice even for the action ESR. Fig. 3 shows the evaluation in favor of the inclusion of an extra id parameter for the ATK action, decreasing the cumulative losses for surprise and (estimated) self-replication with a fraction of the variance of the other choices. The interpretation of this improvement follows from the fact that the social aspects of this soup, predicting others via SUR and providing predictions via (pretend-)ATK, are both helped by the knowledge of the other particles, here, explicitly trained with an identifier unique to each network in the input vector. Initially we hypothesized that the use of a single position for any occurring parameter id would suffice, but, as previously observed by Gabor et al. (2021), using dedicated input parameters proved superior and was thus chosen for the remainder of this work.

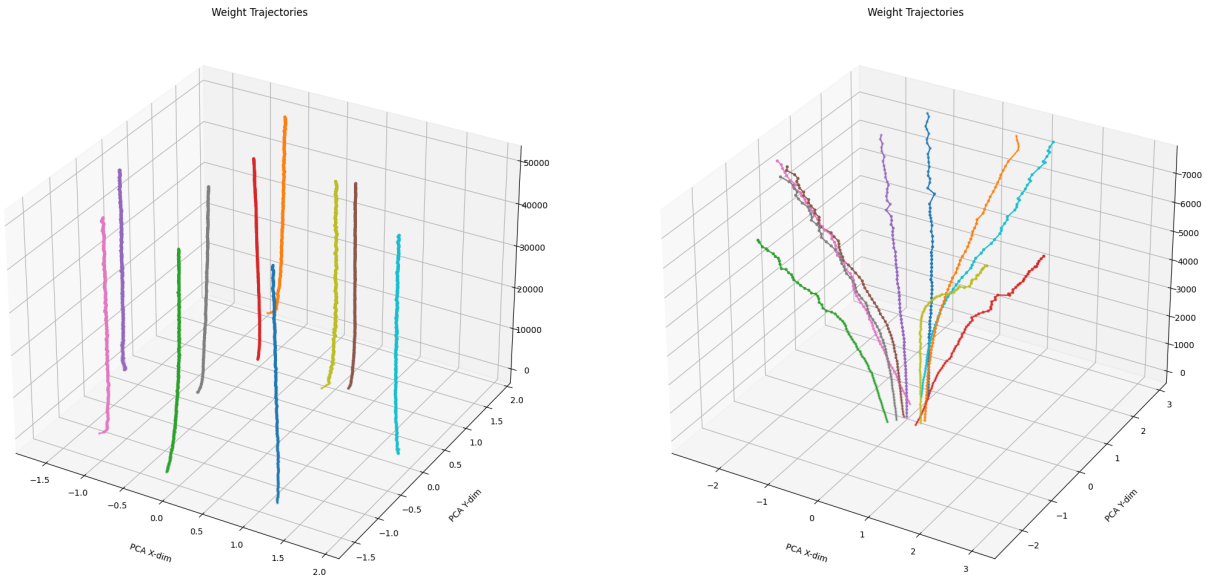

Figure 2: (**left**) This soup consists of 10 network particles $\mathcal{N} : \mathbb{R}^{10} \rightarrow \mathbb{R}$ with one hidden layer of two cells, which are initialized randomly and are interacting solely via the minimize surprise (SUR) action over 50,000 evolution steps. Each particle randomly draws one other network to minimize surprise in every epoch. (**right**) A different soup with 10 similarly constructed particles, which additionally includes the estimate self-replication (SR) (ESR) interaction. In doing so, once per epoch each particle – with a probability of $\gamma = 0.5\%$ – randomly chooses one other network and minimizes the expectation of the partner's current SR loss. The process is run until almost diverged (here at epoch 7000). In these two different settings – due to the resulting difference in the weight values – the scale of the weight space shown here is considerably expanded and not comparable anymore. In both plots we show the weights per network visualized in a two-dimensional weight space based on the transformed X- and Y-axes derived via PCA dimensionality reduction over time.

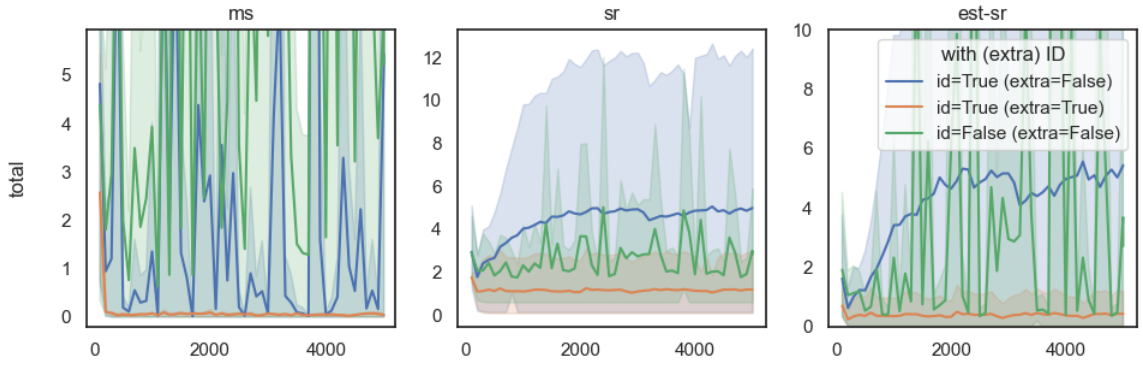

Figure 3: Cumulative total losses of soups with 10 particles each. We test the influence of including an id parameter in the attack (ATK) action to signal the identity of the attacked network. We show the mean and 95% confidence intervals for three different soup runs (with different seeds) for the cumulative surprise (**left**, ms) of each particle for all other particles, the own self-replication loss (**middle**, sr), and the cumulative estimated replication loss (**right**, est-sr), again, of each particle for all other particles (X-axis) over evolution steps (Y-axis). Three cases were compared: (blue), where we include an id parameter but reuse the same input position as the id parameter used for SUR, (orange), where the additional id parameter is placed on a separate input position, and (green), where we do not include the identity of the attacked network. We observe that adding a separate id outperforms the other cases both in terms of loss optimization as well as reducing variance. A catalyst particle was included with attack rate $\alpha = 0.01$.

Using a catalyst to encourage emergent group behavior. To instigate a higher degree of soup stability we observe that we can force all particles to a common sub-region of the weight space, simply by adding a single external particle through which we intend to exert something akin to selection pressure on the soup. In our case the particle's role consists of randomly applying the ATK action on randomly selected networks, forcing convergence or divergence. In doing so, we observe the emergence of more complex behaviors and social structures without the added particle ever changing, which is why we decided on the name 'catalyst'. Similar to the concept of the non-modifiable *beacon* particle shown in Gabor et al. (2021), the catalyst particle is non-modifiable and thus a randomized but constant influence for the particles to consider. Fig. 4 shows the full setting with a catalyst (which is the only attacking particle) and the peer particles, which (besides getting attacked) participate in SUR and ESR interactions with each other. In Fig. 4 (left plot) we can observe the early stages of this process, where singular ATK actions of the catalyst are clearly visible as projections in one sub-space. This 'clustering' into closer proximity then allows the population to better optimize their mutual surprise, resulting in the convergent state observed in later stages of the evolution, depicted in Fig. 4 (right plot). We can also confirm the relation of distance in the weight space to the initial mutual surprise. Fig. 5 shows the heatmap of all particles' mutual surprise losses after the first epoch (including the catalyst), where surprise for, e.g., particles 2 and 3 is considerably higher – which does correspond to the pink and purple particle trajectories in the PCA plot of Fig. 4 (left), which start more distanced from the others. This then also confirms why the 'clustering' ability of the catalyst, which draws the population together, is responsible for the convergence. To reiterate the importance of the catalyst, Fig. 6 shows the direct comparison between the stability of soups with and without a catalyst as indicated by the improved minimization of the various losses (cumulative surprise, self-replication, and estimated SR ability) over the course of the soup evolution. We observe non-negligible variance in both settings, which is mostly explained by the different number and impact of catalyst ATK actions in different random generator seedings. Especially the total surprise loss (Fig. 6, left plot) shows the premature convergence of a soup without catalyst that stagnates to some constant degree of remaining uncertainty without finally converging. The 'clustering' of the ATK actions in contrast do accomplish convergence eventually.

Introducing dynamics into the catalyst. Finally, we explore the idea of training the catalyst itself to observe if and in what way the soup can adjust to resulting dynamics. Fig. 7 shows one iteration of this experiment, where the catalyst very slowly performs self-replication training as is done in related work (which is the only interaction that can be executed by a single particle on its own). To avoid having the SR training immediately perfect the catalyst's predictions (SR training is known to produce SR fixpoints within comparably few iterations; cf. Gabor et al., 2022), we limit the training to occur with the same rate as the ATK actions, i.e., at a chance of 0.1. Over the course of 500,000 evolution steps we observe two phenomena in this dynamic soup:

In the earlier stages of evolution (Fig. 7, left plot) we observe the particles again being 'drawn' towards a similar position in the weight space, but then they follow the adjusted predictions of the catalyst as the SR training modifies its weights and therefore the resulting ATK projections. Remembering how impactful the repeated ATK action is on stationary 'stable' systems, it is remarkable how far and how consistently the catalyst is able to draw the whole soup through the state space without any divergence. The increasing SR ability due to the SR training is certainly a factor that helps with keeping the dynamic movement predictable enough such that the SUR interaction of the individual networks remains learnable without causing divergence.

Later on, we do observe divergent trajectories for all remaining particles except the catalyst in Fig. 7 (right plot), where the soup appears to be 'let go' of the previously consistent navigation through the weight space. We suspect the resulting divergent state is similar to the process occurring in Fig. 2 (right plot), just on a larger scale. By observing the size of the PCA axis in Fig. 7 (right plot) we can conclude that – after the catalyst particle has finished its SR training and the ATK actions produce almost perfect predictions, thus losing any weight displacement ability – the other particles return to their interactions of only training SUR and ESR but this time on peer particles with more complex weight values (and possibly a 'momentum' from the navigated travel of the catalyst training), leading to unexpectedly large losses, large gradient updates, and finally divergence. We reckon that producing long-term stable interactive soups and dynamic processes to navigate such populations safely through the weight space is an interesting topic for future work.

Conclusion

In this work we have adopted the neural network particle soup of Gabor et al. (2019) and extended the approach to rely less on self-replication as the primary component of stability training and more on the social aspect of the soup interactions. We integrated two new actions into the setting: firstly, an estimation of other networks' self-replication ability, based purely on learned memory of the populations unique identifiers; and, secondly, we introduced the concept of minimizing surprise by predicting another network's behavior – given a set of weights – to simulate a trained prevention of influence of a system on the individual. The influence in our soup was exerted in the form of 'selection pressure'-like attack actions. While attacks in previous works were modeled as a (somewhat rare) peer interaction,

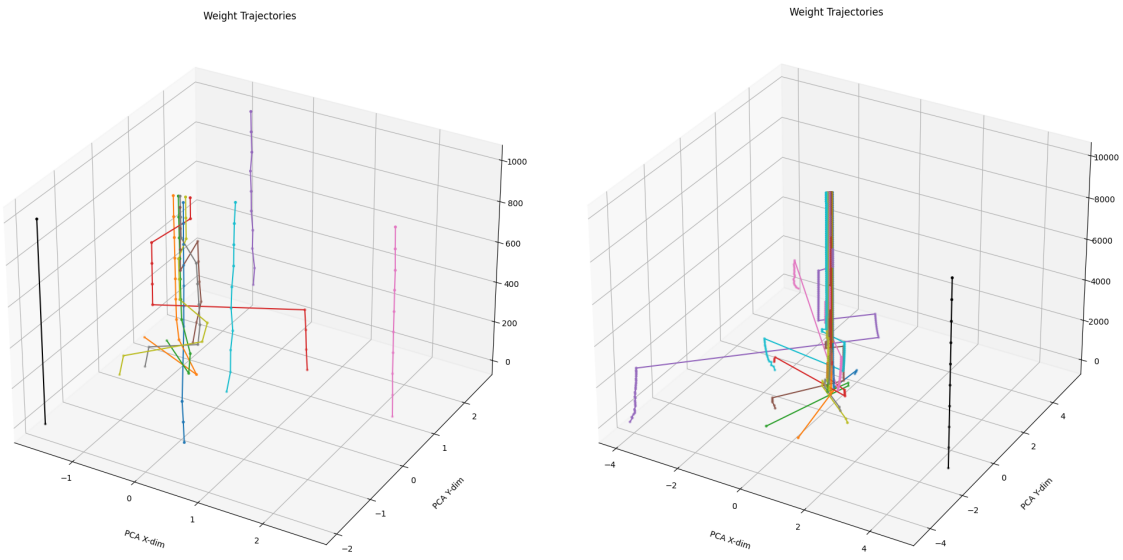

Figure 4: Two perspectives of a soup with 10 network particles $\mathcal{N} : \mathbb{R}^{10} \to \mathbb{R}$ interacting randomly with the actions SUR ($\beta = 1.0$) and ESR ($\gamma = 1.0$). We include a catalyst particle (black trajectory) that does not change over time but simply applies the ATK action randomly once per epoch with a chance of $\alpha = 0.01$. To highlight the early stages of the convergence process we show the evolution once after 1000 epochs (**left**) and a second time after 10,000 rounds of evolution (**right**). All network weights are depicted in the same two-dimensional weight space based on the transformed X- and Y-axes derived via PCA dimensionality reduction.

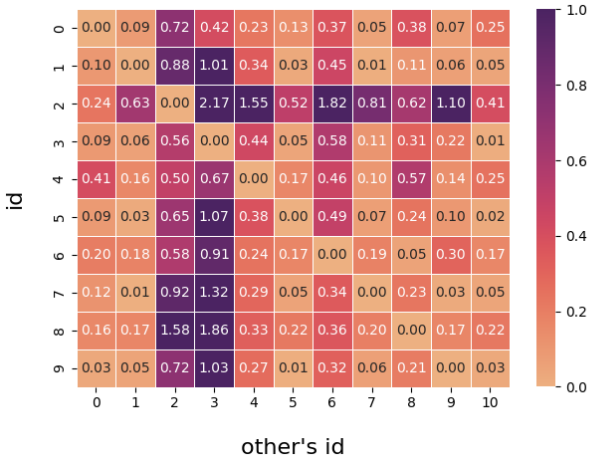

Figure 5: Heatmap of the the mutual surprise losses between a soup of 10 particles (with IDs 0–9) and a catalyst particle (with ID 10, applying the ATK action with $\alpha = 0.01$) after the very first evolution step of the experiment depicted in Fig. 4. The particles most surprised (and surprising) are the ones positioned further away in the weight space during initialization.

in this work we focus on an external catalyst particle that fulfills the role of applying random attacks on the population. While minimal-surprise training alone does provide a form of stability, with the inclusion of the catalyst's attacks we can observe emergent focus of the particles' weights into similar sub-spaces. This behavior leads to improved convergence and minimized self-replication loss as a beneficial side-effect.

The notion of self-replication in our work is strongly influenced by Dawkins (2016) in that it focuses on the ability to simply copy information. We take a first step towards observing other properties associated with self-replication (like metabolism, e.g.) by expanding a particle's feedback loop for self-sufficient development through its peers as we implement surprise minimization in place of self-training. Surprise minimization in uncertain systems may also be helpful for simulating and enforcing interactions between inherently stochastic processes like, e.g., the interaction of quantum particles in the recently emerging field of quantum computing (cf. Nielsen and Chuang, 2010).

In the future we intend to test whether weight *modification* instead of weight *application* as a result of the ATK action could also be used. This would drive the particle setting even more in the direction of hypernetworks (cf. Ha et al., 2016) by training to predict a useful gradient instead of discarding and substituting the whole weight set. This would make singular interactions less dangerous and would allow for more fine-grained weight direction.

We would also like to expand on the catalyst concept and the ability to dynamically influence the population. Future work could include a study on how to model and modify the catalyst so that it remains impactful over the whole evolution

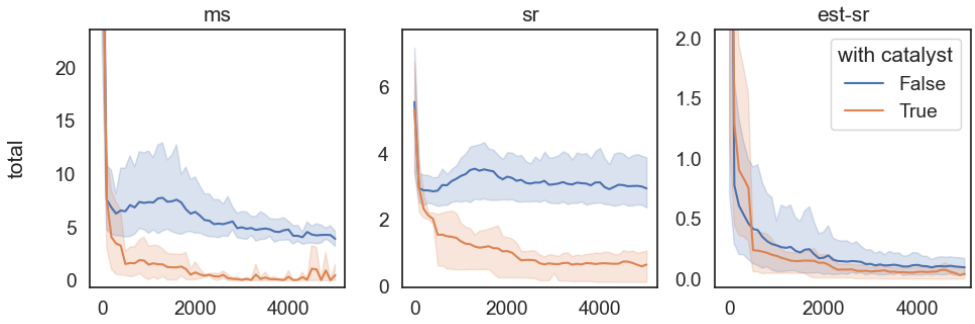

Figure 6: Comparison of cumulative losses of surprise (**left**, ms) of each particle for all other particles, the own self-replication loss (**middle**, sr), and the cumulative estimated replication loss (**right**, est-sr) of all other particles, depending on whether we include the catalyst particle (orange) or not (blue). We show the mean and 95% confidence intervals for three different soup initializations (with three different seeds) up to epoch 5000 on the Y-axis and the evolution steps on the X-axis.

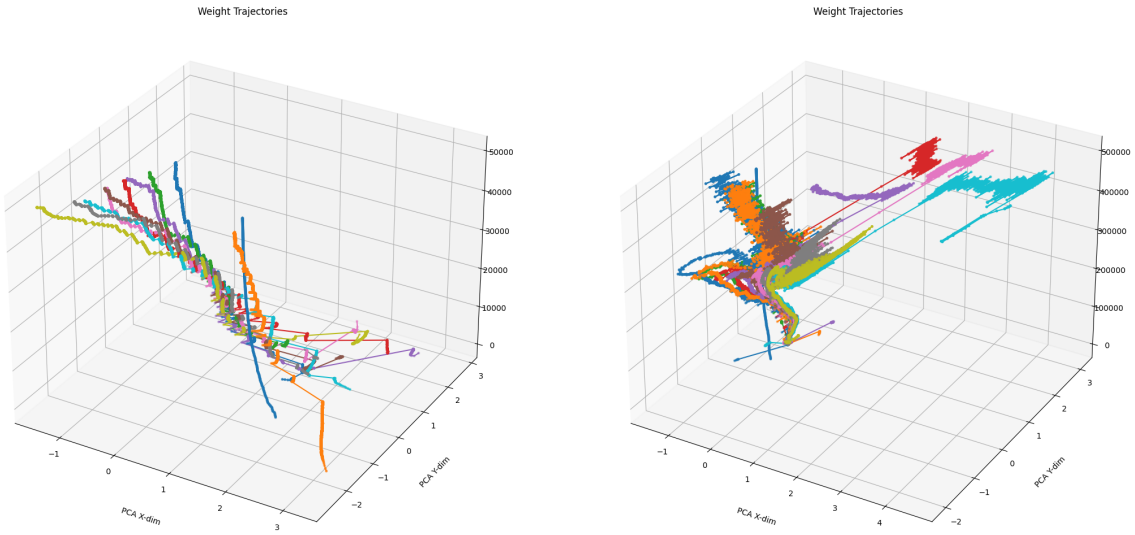

Figure 7: Two perspectives of a soup with 10 network particles $\mathcal{N} : \mathbb{R}^{10} \to \mathbb{R}$ showing the emergent interactions of the actions SUR ($\beta = 1.0$) and ESR ($\gamma = 0.5$) as well as the influence of the catalyst particle ($\alpha = 0.01$, central blue trajectory). In contrast to the setting discussed in Fig. 4, in this case the catalyst indeed changes its weights via the process of self-replication training, indicated by the slightly curved trajectory. The process is shown once in the stage of the soup after 50,000 epochs (**left**) and once much later after 500,000 evolution steps (**right**). Here, the particles attempt to minimize surprise by trying to accommodate the changing projection of the ATK action (of the catalyst), resulting in this moving-cluster pattern observed here. As with the other figure, network weights are depicted in the same two-dimensional weight space based on the transformed X- and Y-axes derived via PCA dimensionality reduction.

while being able to navigate the population, possibly even to manually defined points in the weight space.

Finally, to better understand the soup system as a whole we would like to eventually integrate other components and actions previously defined by Gabor et al. (2021, 2022) and examine the influence of including, e.g., a *beacon* particle or the *learn-from* interaction, which were purposefully omitted in this work as to not obscure the comparatively subtle effect of surprise on the soup. However, combining these actions would allow us to gain further insight on the larger picture of artificial self-organization.

Acknowledgements

This work was partially funded by the *QuCUN* (13N16196) project of the German Federal Ministry of Education and Research (BMBF) funding program "Quantum Technologies – from basic research to market" and partially supported by the German Federal Ministry of Education and Research (BMBF, SCADS22B) and the Saxon State Ministry for Science, Culture and Tourism (SMWK) by funding the competence center for Big Data and AI "ScaDS.AI Dresden/Leipzig".

References

Adami, C. and Brown, C. T. (1994). Evolutionary learning in the 2d artificial life system 'avida'. In Brooks, R. A. and Maes, P., editors, *Artificial Life IV*, pages 377–381. MIT Press, Cambridge, MA.

Adams, R. A., Vincent, P., Benrimoh, D., Friston, K. J., and Parr, T. (2022). Everything is connected: Inference and attractors in delusions. *Schizophrenia Research*, 245:5–22. Computational Approaches to Understanding Psychosis.

Baltieri, M. and Buckley, C. L. (2019). Pid control as a process of active inference with linear generative models. *Entropy*, 21(3).

Berseth, G., Geng, D., Devin, C., Rhinehart, N., Finn, C., Jayaraman, D., and Levine, S. (2020). SMiRL: Surprise minimizing reinforcement learning in dynamic environments. *arXiv preprint arXiv:1912.05510*.

Chang, O. and Lipson, H. (2018). Neural network quine. In *Artificial Life Conference Proceedings*. MIT Press.

Dawkins, R. (2016). *The selfish gene*. Oxford university press.

Der, R. and Martius, G. (2012). *The Playful Machine: Theoretical Foundation and Practical Realization of Self-organizing Robots*. Springer, Berlin, Heidelberg.

Dittrich, P., Ziegler, J., and Banzhaf, W. (2001). Artificial chemistries—a review. *Artificial life*, 7(3):225–275.

Fontana, W. and Buss, L. W. (1994). What would be conserved if" the tape were played twice"? *Proceedings of the National Academy of Sciences*, 91(2):757–761.

Fountas, Z., Sajid, N., Mediano, P., and Friston, K. (2020). Deep active inference agents using monte-carlo methods. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M. F., and Lin, H., editors, *Advances in Neural Information Processing Systems*, volume 33, pages 11662–11675. Curran Associates, Inc.

Friston, K. (2010). The free-energy principle: A unified brain theory? *Nature Reviews Neuroscience*, 11(2):127–138.

Gabor, T., Illium, S., Mattausch, A., Belzner, L., and Linnhoff-Popien, C. (2019). Self-replication in neural networks. In *ALIFE 2019: The 2019 Conference on Artificial Life*, pages 424–431. MIT Press.

Gabor, T., Illium, S., Zorn, M., Lenta, C., Mattausch, A., Belzner, L., and Linnhoff-Popien, C. (2022). Self-Replication in Neural Networks. *Artificial Life*, pages 205–223.

Gabor, T., Illium, S., Zorn, M., and Linnhoff-Popien, C. (2021). Goals for self-replicating neural networks. In *ALIFE 2021: The 2021 Conference on Artificial Life*. MIT Press.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep learning*. MIT press.

Ha, D., Dai, A., and Le, Q. V. (2016). Hypernetworks. *arXiv preprint arXiv:1609.09106*.

Holmes, E., Parr, T., Griffiths, T. D., and Friston, K. J. (2021). Active inference, selective attention, and the cocktail party problem. *Neuroscience & Biobehavioral Reviews*, 131:1288–1304.

Illium, S., Zorn, M., Lenta, C., Kölle, M., Linnhoff-Popien, C., and Gabor, T. (2022). Constructing organism networks from collaborative self-replicators. *arXiv preprint arXiv:2212.10078*.

Kaiser, T. K. and Hamann, H. (2022). Innate motivation for robot swarms by minimizing surprise: From simple simulations to real-world experiments. *IEEE Transactions on Robotics*, 38(6):3582–3601.

Nielsen, M. A. and Chuang, I. L. (2010). *Quantum Computation and Quantum Information: 10th Anniversary Edition*. Cambridge University Press.

Ofria, C. and Brown, C. (1998). The avida user's manual. In Adami (1998), The Avida software is publicly available at ftp.krl.caltech.edu/pub/avida.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. (2019). Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc.

Prokopenko, M. (2013). *Guided self-organization: Inception*, volume 9. Springer Science & Business Media.

Randazzo, E., Versari, L., and Mordvintsev, A. (2021). Recursively fertile self-replicating neural agents. In *ALIFE 2021: The 2021 Conference on Artificial Life*. MIT Press.

Ritz, F., Ratke, D., Phan, T., Belzner, L., and Linnhoff-Popien, C. (2021). A sustainable ecosystem through emergent cooperation in multi-agent reinforcement learning. In *Artificial Life Conference Proceedings 33*, volume 2021, page 74. MIT Press, One Rogers Street, Cambridge, MA.

Serugendo, G. D. M., Gleizes, M.-P., and Karageorgos, A. (2005). Self-organization in multi-agent systems. *The Knowledge engineering review*, 20(2):165–189.

Van den Bergh, F. and Engelbrecht, A. P. (2000). Cooperative learning in neural networks using particle swarm optimizers. *South African Computer Journal*, 2000(26):84–90.