A Simple Sparsity Function to Promote Evolutionary Search

Clifford Bohm[4,2] and Arend Hintze[2,3] and Jory Schossau[1,2]

[1]Michigan State University, Department of Computer Science and Engineering
[2]Michigan State University, BEACON Center for the Study of Evolution in Action
[3]Dalarna University, Department of MicroData Analytics
[3]Michigan State University, Department of Integrative Biology
cliff@msu.edu

Abstract

This study investigates the relationship between sparse computation and evolution in various models using a simple function we call *sparsify*. We use the sparsify function to alter the sparsity of arbitrary matrices during evolutionary search. The sparsify function is tested on a recurrent neural network, a gene interaction matrix, and a gene regulatory network in the context of four different optimization problems. We demonstrate that the function positively affects evolutionary adaptation. Furthermore, this study shows that the sparsify function enables automatic meta-adaptation of sparsity for the discovery of better solutions. Overall, the findings suggest that the sparsify function can be a valuable tool to improve the optimization of complex systems.

Introduction

Evidence of evolution is a common hallmark in natural systems. One of the more profound results of evolution seems to be the distribution of connectivity in evolved networks. Similarities in connectivity have been observed in gene regulatory networks (Foster et al., 2006; Quayle and Bullock, 2006), neural connectivity (Eguiluz et al., 2005), and even food (Estrada, 2007; Otto et al., 2007) and social networks (Barabâsi et al., 2002). Power-law distributions are often observed in these contexts, and a common conclusion is that power-law distributions are a strong indicator of past evolution (Barabasi and Oltvai, 2004).

For example, Power-law distributions can emerge as the consequence of node duplications (or gene duplications in natural systems (Wagner, 2003)) and preferential attachment (Barabási and Albert, 1999). Because there is such a strong correlation between power-law distributions and evolutionary adaptation, it has been suggested that in order for complexity to evolve that the system must be in a critical state — at the edge of chaos — and that this state is facilitated by networks having a distribution of connectivity classified as a power-law distribution. Researchers either try to enforce this distribution, or wait for it to emerge from evolutionary processes (Hintze and Adami, 2008). However, there may be less causation than correlation for the power-law distribution as a necessary driving force for evolving systems. There exists criticism about the notion that biological networks are following a power-law distribution (Khanin and Wit, 2006; Lima-Mendez and Van Helden, 2009), and others questioning the idea that preferentially attached connections (Barabási and Albert, 1999; Bhan et al., 2002; Ravasz and Barabási, 2003; Foster et al., 2006) are needed to create these power-law distributions and are the driving factor behind these topologies.

Unfortunately, creating networks with power-law distributions is a computationally costly endeavor (Feng et al., 2022), with reliable methods often requiring a slow process of pruning (Liu et al., 2017; Frankle and Carbin, 2018; Lin et al., 2019).It may simply be that there are other mechanisms that do not directly create power-law distributions, but are equally good for evolvable systems and are less computationally costly. We propose that evolution of sparsity is such a mechanism that is easier to enforce than power-law distributions and is beneficial for evolving systems.

Ideally we would like to explore sparsity, outside the context of power law distributions, as a possible independent driver of evolutionary success. To do this, we would need to control distribution degree and sparsity independently. However, it is not clear how to separate these two intrinsically linked effects. Here, we show that simply biasing sparsity — whether or not it creates power law distributions — is enough to drive evolutionary adaptation. Furthermore, the systems we are examining may not be large enough to determine if they exhibit a power-law distribution. Nonetheless, we will demonstrate that even in the absence of a power-law distribution, sparsity alone can significantly impact the system. Thus, the inquiry of whether sparsity leads to power-law distributions will remain unanswered.

In (Bohm et al., 2022), the authors used a transformation function to establish sparsity in recurrent neural networks. Bohm, et al. used this function — we refer to as the *sparsify* function — to bias the conversion from genomes to a computational model such that weights with a value of 0 appeared more often than usual. Here, we use the sparsify function to investigate how the level of sparsity affects evolution across various models. The sparsify func-

tion takes five values: x, min, t_1, t_2, and max (where $min \leq t_1 \leq t_2 \leq max$). The function transforms values of x less than min to -1.0, between t_1 and t_2 to 0.0, and greater than max to 1.0. Meanwhile, values of x between min and t_1 and between t_2 and max are transformed into the ranges $[-1.0 \ldots 0.0]$ and $[0.0 \ldots 1.0]$ respectively (see Equation 1 and Figure 1).

$$s(x, min, t_1, t_2, max) = \begin{cases} -1.0 & x \leq min \\ \frac{(x-min)}{(t_1-min)} - 1 & min < x < t_1 \\ 0.0 & t_1 \leq x \leq t_2 \\ \frac{(x-t_2)}{(max-t_2)} & t_2 < x < max \\ 1.0 & x \geq max \end{cases} \quad (1)$$

In this work, we are interested in how different levels of sparsity affect evolution, so we only use the sparsify function to control the ratio of zeros to non-zeros while maintaining an equal ratio of negative and positive outputs. In particular, we use values of x in the range [min ... max] and set the other values such that $(t_1 - min) = (max - t_2)$. For example, $min = 0$, $t_1 = 0.25$, $t_2 = 0.75$, and $max = 1$, which results in half of all values being zeros, or a sparsity of 0.5. Or, $min = 0$, $t_1 = 0.45$, $t_2 = 0.55$, and $max = 1$, which generates a sparsity of 0.9. In the remainder of the paper, when we use "initial sparsity" we are indicating that an experiment was conducted using a set of values to the sparsify function so that it would generate the given ratio of zeros. Note that after this initial generation of genomes, the process of evolution may alter the distribution of genomic values such that values drawn from the sparsify function no longer reflect the initial sparsity, and thus sparsity may be affected through mutation. In this way, the sparsity may evolve.

It is also possible to use the sparsify function to alter the ratio of extreme values (i.e., -1.0 and 1.0) by allowing values outside the range $[min \ldots max]$. Allowing out-of-range values would increase the ratio of extreme values (-1 and 1), but an investigation of this effect is outside the scope of this study.

Using the *sparsify* function, we show how evolutionary search can be positively affected. Moreover, our results demonstrate that the sparsify function allows automatic meta-adaptation of sparsity for the discovery of more optimal solutions.

Materials and Methods

To test the effect of the *sparsify* function on evolution, we chose two different problem domains for 4 total tasks, using 3 types of evolving structures:

- A changing NK-fitness landscape domain that we used to evolve two structures that generate bit string solutions.

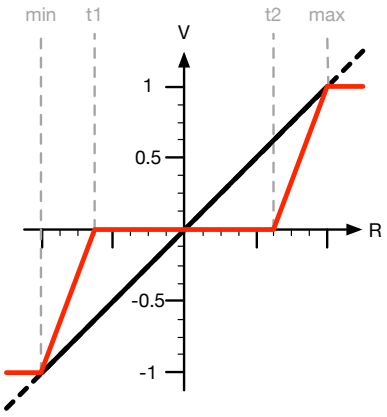

Figure 1: Illustration how the *sparsify* function translates a value R (black line) into a new value V (red line), see Equation 1. In the experiments described below the input values used in the sparsify function are limited to $[min \ldots max]$ as indicated by the solid black versus dashed black line.

  - Indirect Encoding: a simple indirect encoding model that uses sparsify to bias a conversion matrix.
  - GRN: a gene regulatory network model that uses sparsify to bias gene interactions.

- A behavioral task domain that we used to evolve the structure of recurrent neural networks (RNN) that use sparsify to bias their weights matrices. The RNNs were evolved on three behavior tasks.
  - Maze: An navigation and memory task.
  - Berry: An foraging task with punishment for greedy algorithms.
  - Blind Navigation: An algorithm task involving no sensorial inputs, but requiring counting.

Changing NK-Fitness Landscape

The NK-fitness landscape (Kauffman and Levin, 1987) model is a well-established benchmark (Pitzer and Affenzeller, 2012) to test genetic algorithms and to study evolutionary dynamics (Altenberg, 1996). In this model, a phenotype is defined as a vector of N binary values (traits), and the degree to which traits interact is defined by the value K. When, for example, $K = 1$, each trait interacts with one other trait, conventionally the following trait in the vector. The phenotype vector is structured as a ring, such that the rightmost traits can interact with the leftmost. The fitness for each phenotype is defined by a fitness table, that defines the fitness contribution for every configuration of each trait and its K associated traits. This fitness lookup table is of size $2^K \times N$, and is filled with random numbers from the interval $(0.0 \ldots 1.0]$ (this is excluding zero). The fitness (W) of each phenotype is the product of the contributions from each set of K associated traits W_i:

$$W = \left(\prod_{i=1}^{N} W_i \right)^{1/N} \quad (2)$$

.

As K defines the degree of interactions between traits, it effectively controls the ruggedness of the landscape. Observe that in the case of $K = 0$ (no trait interaction) the landscape is smooth with a single peak, and in the extreme case of $K = N - 1$, the landscape is completely random.

Traditionally, the lookup table is static. Here, the NK-fitness landscape is converted to a dynamic fitness function by making the values in the lookup table depend on the current generation and a speed factor (V); for more details, see Mehra and Hintze (2022).

Moreover, in most examples of NK-fitness landscapes, point mutations change individual traits by flipping the bit defining them. In that case, the genotype is identical to the phenotype — i.e., a direct encoding. However, there is no restriction on the process by which the phenotype bit strings are generated. We test two different encodings to generate bit strings that allow us to look at two applications of the sparsify function. In both cases, we evolved populations of size 100 for $20,000$ generations. At every generation, we used roulette wheel selection (Miller et al., 1995) to identify the parent organisms and used these to produce mutated offspring for the next generation (see below for details on the frequency and types of mutations). In all NK-fitness landscape experiments, we used $N = 20$, while the ruggedness K and the speed V of the model were experimentally varied. All combinations of $V = 0.0, 0.0001, 0.01$ and $K = 0, 1, 3, 5$ were tested with each initial sparsity from $[0, .1, .3, .5, .7, .9, .95, .99]$. 300 replicates of each experiment were performed.

In the results for the changing NK-landscape, we show the mean Hamming distance to the highest current fitness peak (based on the current state of the dynamic look-up table). From this perspective 0 would indicate that the best possible solution was discovered and values > 0 indicate a degree of mismatch from the optimal solution. This does require that we evaluate every possible solution.

Indirect Encoding

In the indirect encoding (IE), each agent is individually defined by both a genome (G) with N real values in the range $[-1.0 \ldots 1.0]$ and a connectivity matrix (M) with real values in the range $[-1.0 \ldots 1.0]$ of size $N \times N$. Both the genome and the connectivity matrix are initialized with values drawn from the distribution resulting from the sparsify function. To evaluate an agent, we first convert the genome into a new phenotype vector (P) (see Equation 3) and then discretize the values in the resulting vector P such that values $\leq 0.0 = 0.0$ and values $> 0.0 = 1.0$ in order to generate an NK-fitness bit string.

$$P = G \cdot M \quad (3)$$

Mutations in the indirect encoding are performed at a rate of $\mu = 0.001$ per genome site and connectivity matrix site, and when they occur, they replace a value in the genome with a new value drawn using the sparsify function.

Gene Regulatory Network

The second method that we use to generate bit stings for the NK-fitness model is a gene regulatory network. In this developmental encoding (Hintze et al., 2020) model, evolution is used to define an interaction network that is processed over time to establish a solution. In this model, agents have an interaction matrix C, of size $N \times N$, that determines how each of their N genes will influence one another — i.e., the degree to which they up or down regulate one another. The values in C matrix are seeded with the sparsify function.

In order to evaluate an agent, each of that agent's gene expression levels are set to $E_i = \frac{1}{N}$. The agent's levels are updated 100 times, allowing for a change in gene expression as defined by the interaction matrix. Over the 100 time steps the updated gene expression level E'_i for every gene i is computed as follows:

$$q_i = \sum_{j=0}^{N} E_j C_{i,j} \quad (4)$$

$$Q_i = \frac{2.0}{1.0 + e^{-\alpha q_i}} - 1.0 \quad (5)$$

$$E'_i = E_i - \beta(E_i - Q_i) \quad (6)$$

In Equations 4, 5, and 6 $\alpha = 1.0$ and $\beta = 0.2$, where α simulates the temperature of the system, and β a dampening factor. α and β are treated as constants, and remain unchanged throughout the experiment.

After 100 updates, the values in the expression levels vector, E, are discretized as in the IE and the resulting bit sting is evaluated on a changing NK-fitness landscape.

Mutations to the GRNs change values in the interaction matrix, at a rate of $\mu = 0.001$ per site, to new values drawn from the sparsify function.

Since the expression levels of each gene E_i do not reflect concentrations and thus do not need to sum to 1.0. Instead, they signal levels of over- or under-expression in the range of $[-1.0 \ldots 1.0]$.

Recurrent Neural Network

In addition to the NK-fitness analysis describe above, we tested sparsify in the context of evolved recurrent neural networks (RNN) tasked with steering virtual robots in three resource collection tasks.

Our RNNs are comprised of a single layer of perceptron gates. The number of sensory input values RNNs receive and of motor outputs they deliver differed between

tasks. Recurrence is implemented through additional input and output values — here referred to as "hidden" as they are not visible to anything outside the RNN. As such, there are "sensory input" + "hidden" number of total input values fed into the network and "motor output" + "hidden" total output values generated by the network. Each gate has a weighted connection from each input node and an unweighted connection to a dedicated output node. Weights are drawn from a genome that is seeded randomly, and the *sparsify* function is used to transform genomic values to the weights in this representation. The $\tanh$ function was used as the threshold function ($[-1.0 \ldots 1.0]$). In addition, each gate included a bias ($[-1.0 \ldots 1.0]$). As the tasks require binary values for output, outputs generated by the RNNs are discretized so that values ≤ 0 are transformed to 0 and values > 0 are transformed to 1.

In all the RNN results, the populations are size 100. Tournament selection (tournament size 10) is used to select parents who reproduce asexually. Offspring receive mutations: point mutation at a rate of 0.01/site; copy mutation at a rate of 0.0002/site; and deletion mutations at a rate of 0.0002/site. See the replication notes for more details. We ran 155 replicates of each task for each initial sparsity from $[0.0, 0.1, 0.3, 0.5, 0.7, 0.9, 0.95, 0.99]$.

Task: Berry World

In Berry World, an agent is placed on an 8×8 grid surrounded by a wall, where each grid location contains a berry. The berries come in two types and agents receive one point every time they consume a berry. If, after consuming a berry, an agent moves away from an empty location, the location is filled with a berry of a random type. Each time the agent consumes a berry type different from the previous one, then they incur a cost for switching (1.75 points). Agents have 400 time steps to collect resources and maximize their points. During each time step, an agent can turn left or right (45°), move forward, collect a berry, or take no action. The agent's sensory array is indicated in figure 2(a). Agents have three binary outputs, where 100 codes for left, 010 for right and, 110 for forward. If the third output is 1, then the agent collects the berry at its current location and the other outputs are ignored. To achieve high fitness, an agent must collect numerous berries while also minimizing transitions between berry types. We have identified a number of strategies that generally require some combination of search, navigation, and memory about the last type of berry collected.

Task: Blind Navigation

In the Blind Navigation task, the agent is placed on an 8×8 toroidal grid (i.e., the edges wrap). The agent is provided 80 time steps to navigate to every location on the gird, but they have no sensory input, so in order to achieve perfect fitness they must implement some form of counting (i.e., memory and cycles). On each time step, agents can turn left or right (45°), move forward, or take no action (i.e., the outputs are configured like Berry world without the third output). See figure 2(b) for an illustration of this task.

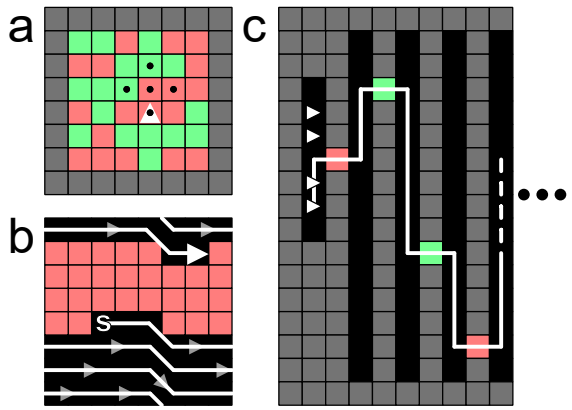

Figure 2: Illustrations of tasks used to evolve RNNs. a) Berry World, where gray indicates wall, green and red the two food types. The agent is indicated by a white triangle (in this case, facing north). The black dots indicate the locations visible to the agent. b) Blind Navigation, where red indicates an unvisited location, black a visited location. Here, the agent (the white triangle) has partially competed the task, having started at the start marker (s). The white line indicates the path taken by this agent. c) A partial maze of the type used in the Maze task. The maze displayed here continues to the right. Gray indicates wall. Red and Green represent cues that the next opening will be to the left or to the right from the door, respectively. The triangles indicate starting positions (determined randomly) and the white trace shows the path of a perfectly performing agent.

Task: Maze

This maze-solving task was previously published in (Edlund et al., 2011). The task places the agent on one side of an enclosed area divided into hallways by walls. In each wall, there is a single opening, and in each opening is a cue indicating if the next opening will be left or right of the agent's current location (See figure 2(c)). Agents can sense cues at their current location as well as if there is a wall directly left, right, or in front of them. Agents have two binary outputs where 10 codes for left, 01 for right, 11 for forward, and 00 for no action. Agents accrue fitness for each opening they pass though. The agent has a limited number of time steps in each maze, so they must attend to the cues in order to achieve high fitness. This task requires the agent to perform simple navigation and to remember at least one bit of information (i.e., the cue in the last opening they passed through).

Results

We tested the effect of different initial sparsities (0.0, 0.1, 0.3, 0.5, 0.7, 0.9, 0.95, and 0.99) in the context of three evolving computational structures using four tasks from two problem domains.

changing NK-fitness landscape results

Two of the evolving structures were tested using the changing NK-fitness landscape-based model, where we investigated three different speeds of change $V = (0.0, 0.0001, 0.01)$ and four levels of ruggedness $K = (0, 1, 3, 5)$.

Figure 3 shows Δ_{peak}, which is the distance to the highest peak in the fitness landscape at generation $20,000$ averaged across each replicate for the given condition. We find that the best observed outcomes tend to be associated with an initial sparsity of 0.9. This is not the case for the completely smooth and static landscape ($K = 0$ and $V = 0.0$) where the highest levels of initial sparsities adapt optimally. But for rugged landscapes, optimal scores are found at lower initial sparsity. Moreover, the differences between different initial sparsity < 0.9 on rugged landscapes were small, while overly sparse matrices > 0.9 resulted in less optimal outcomes. The differences in the results of the IE and GRN encodings are minimal, with the IE model often finding slightly better solutions. The optimal sparsity is often the same or similar for both encodings.

Interestingly, we find that the sparsity at the end of the evolutionary period ($20,000$ generations) sometimes differs from the initial sparsity bias (see figure 4). For initial sparsity < 0.9 we find little change between the initial and the final evolved sparsity. But for initial sparsity $>= 0.9$ there is a significant reduction in the amount of sparsity by the end of the evolutionary process (i.e., an increase in the number of non-zero values).

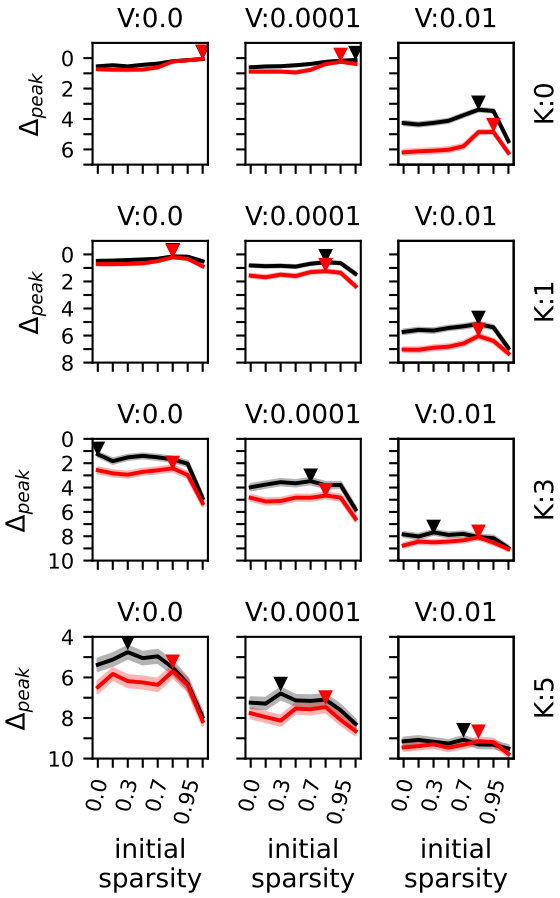

Figure 3: Mutational distance of evolved solutions to the highest fitness peak. Results show the mean distance to the peak (Δ_{peak}) of 300 independently evolved populations, for four different values of K (0, 1, 3, and 5; panel rows) and for three different speeds V in which the environment was changing (0.0, 0.0001, and 0.01; panel columns). In black, the results for the indirect encoded solutions (IE) and in red, the results for the GRN. Shadows behind the lines show the standard error. The highest attained fitness for each parameter combination ($S_{initial}$ x-axes) is highlighted by a triangle.

RNN results

In the changing NK-fitness experiments, the sparsify function was used as part of the process to generate solutions that were then evaluated to generate a fitness score. In the RNN experiments, the RNNs were evolved to solve behavior tasks where fitness evaluation required agents to interact with environments (or at least to act within environments) over multiple time steps. In these tasks, the agent's actions affect the state of the environment and to achieve high fitness the agents must respond to these changes. In some of the environments, the agents also needed to remember environment states or maintain complex internal states. In other words, where the NK-fitness model experiments use the sparsify function to create a static solution. In the RNN models, the sparsify function is used to create dynamic cognitive devices that can maintain internal states while they interact with the task environment.

The adaptation of an RNN to different environments was tested with different values for initial sparsity (0.0, 0.1, 0.3, 0.5, 0.7, 0.9, 0.95, 0.99) and with different numbers of hidden nodes (4, 8, 16, and 32). Figure 5 (a), shows how the RNNs performed on each task, where the x-axis is the initial sparsity and the line colors represent the number of hidden nodes. We find that RNNs with a lower number of hidden nodes (4, 8, and 16) perform well at low initial sparsity, and worse as the sparsity increases. On the other hand, while RNNs with 32 hidden also suffer at very high initial sparsity, they also suffer at low sparsity. In fact, they appear to preform best with a sparsity around 0.9 — and at this level of

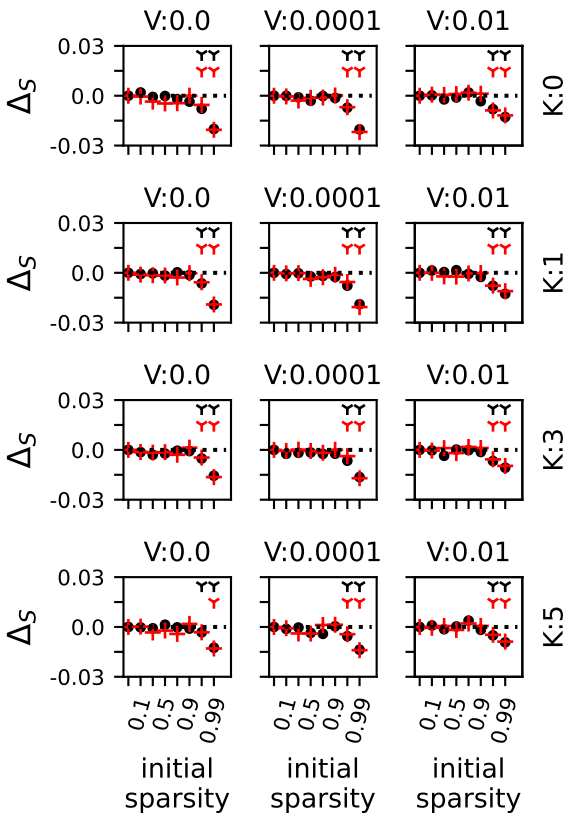

Figure 4: Deviation between the initial sparsity (dotted line) and the final sparsity of evolved solutions for different parameters of K (ruggedness), V (speed of change), and $S_{initial}$. The deviation Δ_S between the initial sparsity $S_{initial}$ (x-axes) is shown for four different values of K (0, 1, 3, and 5; panel rows) and for three different speeds V in which the environment was changing (0.0, 0.0001, and 0.01; panel columns). Black dots represent results from the indirect encoding (IE) and in red the results for the GRN encoding. The red and black Y markers indicate where the results are significantly different from the neutral expectation (dotted line) using a Mann Whitney U test with significance as a p-value $< 10e-4$ (to account for multiple hypothesis testing).

sparsity their performance rivals — and in some cases even exceeds — the performance of smaller RNNs with the same initial sparsity.

In figure 5 (b) the average sparsity of the highest performer by task, number of hidden nodes, and initial sparsity for each replicate. RNNs have a greater ability to evolve away from the initial sparsities than the IE or GRN models. For example, networks with an initial sparsity of 0.7 in the blind navigation task evolved to an average sparsity of 0.45.

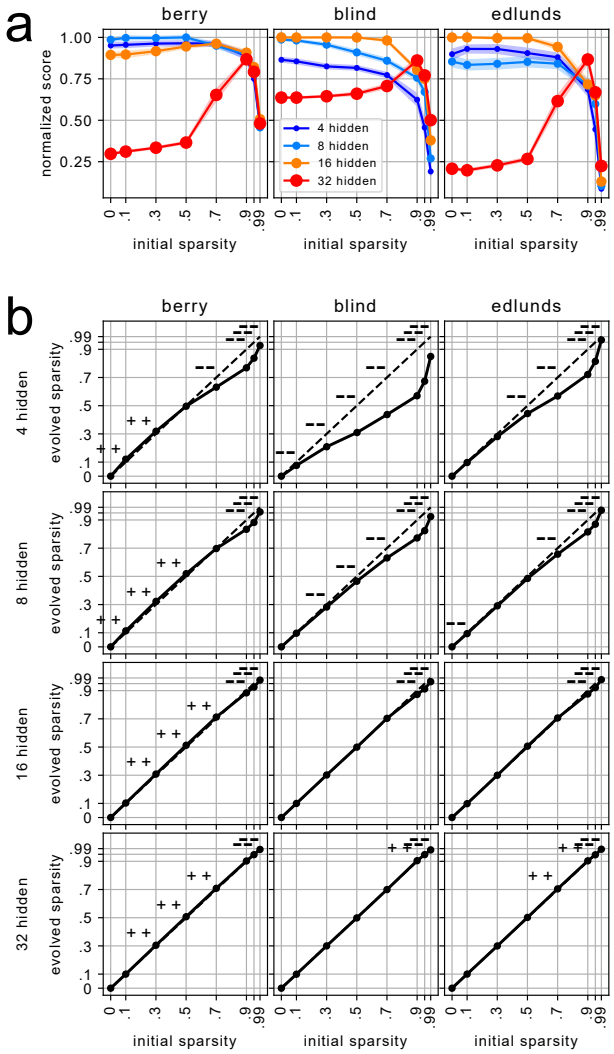

Figure 5: Results from recurrent neural networks evolved to solve the Berry World, Blind Navigation, and Maze tasks. 155 replicates were run for each condition (task x hidden x initial sparsity). a) shows average maximum normalized scores after 20k generations of evolution using different numbers of hidden nodes (4, 8, 16, or 32 indicated by shades of blue and red) and initial sparsity (x-axis). The shaded areas show 95% confidence intervals. b) shows evolved sparsities vs initial sparsities for the conditions in (a). Each subplot shows a task and a number of hidden nodes. The solid black lines show the average sparsity of the highest scoring agents after $20,000$ generations of evolution. The dashed black line shows the average sparsity of the initial populations for reference (i.e., the neutral expectation if evolution did not affect sparsity). The $++$ and $--$ markers indicate where the evolved results are significantly different from the initial results using a Mann Whitney U test with a p-value $< 10e-4$.

Discussion and Conclusion

Biological networks have sparse connections; Genes usually interact with only a few other genes, and neurons in a brain or nervous system only wire to a subset of other neurons. Sparsity is even present in social and ecological interaction networks. A great deal of literature has investigated a particular type of sparsity where the connectivity between nodes can be described with a power-law distribution and furthermore that networks with this property exhibit "edge-of-chaos" dynamics, which enhances evolutionary outcomes. We do not oppose these assumptions, but are interested in the effect of sparsity itself (that is, random sparsity that does not follow a power-law distribution) on evolutionary processes.

Here, we consider sparsity generated by a simple function that transforms values from one range to values in a new range where zeroes are more common. This function is used in contexts where these zero values will cause broken connections, such that an overabundance of zeros will generate sparsity. We test if and when this function can be leveraged to improve the evolutionary search of a GA. Typical improvements to a GA would enforce a power-law distribution of the network connections or implement a computationally costly preferential attachment algorithm. Instead, we use a simple function we call *sparsify* (see Equation 3 and Figure 1).

When testing the effect of this function for different computational models and on different optimization problems, we can confirm two outcomes.

- Given the right degree of sparsity, evolutionary search is improved over a wide range of systems and problems.

- Agents using the *sparsify* function will evolve to have actual sparsities adapted to the needs of the environment. That is, while agents initially have one level of sparsity that is the result of random initialization and the sparsify function parameters, they may evolve to be either more sparse or less sparse after adaptation.

When the initial sparsity was set to 0, we did not observe any changes. This is because at 0 sparsity the sparsity function has no effect, as the probability of generating a zero weight by mutation is very low (about 1 in 2^{64}), equivalent to not using sparsity function. On the other hand, when the sparsity is set to a very high value (.99), the lack of resulting connectivity between nodes hinders the discovery of any solutions. However, the probability of generating a zero weight depends on the sparsity function (i.e., the initial sparsity in each condition), and at intermediate sparsity levels, the odds of a mutation generating a zero can be quite high. We found that this is sufficient to allow evolution to tune values to affect altered sparsities, resulting in the offsets seen in Figure 5(b).

The RNN matrices can be considered as a multi-parameter or multi-objective problem where weight values work together to generate high-quality solutions. If the number of parameters is too small, then there may not be enough values to allow for synergy, and the network will be underparameterized and unable to solve certain problems. Conversely, if there are too many parameters, evolution cannot effectively search the parameter space within a reasonable time limit. For example, a single layer perceptron cannot solve the XOR classification problem due to a lack of complexity, but adding multiple layers can enable a solution Murtagh (1991). Similarly, a large number of network connections can quickly grow the search space beyond what is feasible for computational search using evolution De Jong (1988); Stanley et al. (2019).

Our results support the conjecture that another way to achieve a network that is effectively too small for meaningful computation is to create a network that is too sparse despite its number of parameters. Similarly, a network that is very large and not sparse would create an explosion of parameters, encumbering evolutionary search when time is limited. So the interplay between sparsity and network size seems to be an interesting area for future investigation. Our results show that a population of networks that are the right size will successfully evolve weights to solve the task. Meanwhile, a population of networks that are too small will be unable to evolve as good a solution and a population of networks that are too large, should be able to evolve if they can adapt their sparsity to fit the problem. Because we do not know how many parameters are ideal for each task to see such an effect, then we tested several different network sizes for the RNN model.

In addition to testing the RNNs with different initial sparsities, we vary the number of parameters as the number of hidden nodes. This allowed us to investigate an additional dimension that was not available in the indirect or developmental encodings. We first identified that in the absence of sparsity there was an optimal number of hidden nodes for each task, indicated in figure 5(a). We found that while the RNNs with large hidden counts tended to underperform, they "recovered" their fitness when the sparsity was high, but not so high as to effectively cause the model to be underparameterized again. We hypothesize that in this way, sparsity acts as an evolvable corrective force for overparameterization.

There is a similar recovery observed when measuring initial sparsity vs. evolved sparsity 5(b). In larger networks, higher initial sparsity appears to have little effect on evolution for final sparsity. This may be the effect of several possible causes:

- That there may be RNN nodes that are not being used in the computation. The weights on these nodes would not be influenced by selection, and so they would tend

to the ratio established by the distribution defined by the sparsify function.

- It may be that there are more configurations in larger networks, so the probability of finding a working solution with a distribution of weights closer to the initial values is more likely.

We did not explore all possibilities that the *sparsify* function offers, and only focused on symmetric parameterization of the function (i.e., where the ratio of positive numbers to negative numbers was 1.0). However, the range of non-zero numbers can be easily biased with the correct set of parameters. This would create a matrix with mostly positive or negative weights. If such a bias was applied to an GRN's interaction matrix, it would result in a bias to the ratio of gene production versus inhibition. Similarly, we limited inputs to the sparsify function so that they were between min and max. Relaxing this restriction, would generate matrices where extreme values were unusually common. If applied to the weights of an RNN, this would result in more discretized information flow.

The *sparsify* function has proven successful in enabling adaptation in three different systems across four different problems. Therefore, it is reasonable to assume that it can also improve other systems applied to different problems or using different optimization methods. However, determining the appropriate thresholds and network sizes for each individual problem will remain a challenge. It is also possible that sparsity may have a negative effect on certain systems or tasks, particularly in situations where more domain knowledge is available to aid in the search. It is important to note that the No Free Lunch theorem still holds true (Wolpert and Macready, 1997; Hintze et al., 2019).

Acknowledgements

The computations were performed on resources provided by SNIC through Uppsala Multidisciplinary Center for Advanced Computational Science (UPPMAX) under Project SNIC 2020-15-48 and by the Institute for Cyber-Enabled Research at Michigan State University. This work was in part sponsored by the BEACON Center for the Study of Evolution in Action NSF Cooperative Agreement No. DBI-0939454.

References

Altenberg, L. (1996). B2. 7.2 nk fitness landscapes. *Evolution*, 2.

Barabási, A.-L. and Albert, R. (1999). Emergence of scaling in random networks. *science*, 286(5439):509–512.

Barabâsi, A.-L., Jeong, H., Néda, Z., Ravasz, E., Schubert, A., and Vicsek, T. (2002). Evolution of the social network of scientific collaborations. *Physica A: Statistical mechanics and its applications*, 311(3-4):590–614.

Barabasi, A.-L. and Oltvai, Z. N. (2004). Network biology: understanding the cell's functional organization. *Nature reviews genetics*, 5(2):101–113.

Bhan, A., Galas, D. J., and Dewey, T. G. (2002). A duplication growth model of gene expression networks. *Bioinformatics*, 18(11):1486–1493.

Bohm, C., Albani, S., Ofria, C., and Ackles, A. (2022). Using the comparative hybrid approach to disentangle the role of substrate choice on the evolution of cognition. *Artificial Life*, 28(4):423–439.

De Jong, K. (1988). Learning with genetic algorithms: An overview. *Machine learning*, 3:121–138.

Edlund, J. A., Chaumont, N., Hintze, A., Koch, C., Tononi, G., and Adami, C. (2011). Integrated information increases with fitness in the evolution of animats. *PLoS computational biology*, 7(10):e1002236.

Eguiluz, V. M., Chialvo, D. R., Cecchi, G. A., Baliki, M., and Apkarian, A. V. (2005). Scale-free brain functional networks. *Physical review letters*, 94(1):018102.

Estrada, E. (2007). Food webs robustness to biodiversity loss: the roles of connectance, expansibility and degree distribution. *Journal of theoretical biology*, 244(2):296–307.

Feng, F., Hou, L., She, Q., Chan, R. H., and Kwok, J. T. (2022). Power law in deep neural networks: Sparse network generation and continual learning with preferential attachment. *IEEE Transactions on Neural Networks and Learning Systems*.

Foster, D. V., Kauffman, S. A., and Socolar, J. E. (2006). Network growth models and genetic regulatory networks. *Physical Review E*, 73(3):031912.

Frankle, J. and Carbin, M. (2018). The lottery ticket hypothesis: Finding sparse, trainable neural networks. *arXiv preprint arXiv:1803.03635*.

Hintze, A. and Adami, C. (2008). Evolution of complex modular biological networks. *PLoS computational biology*, 4(2):e23.

Hintze, A., Hiesinger, P. R., and Schossau, J. (2020). Developmental neuronal networks as models to study the evolution of biological intelligence. *Workshop on neuro-evo-devo at the 2020 Conference on Artificial Life*.

Hintze, A., Schossau, J., and Bohm, C. (2019). The evolutionary buffet method. *Genetic programming theory and practice XVI*, pages 17–36.

Kauffman, S. and Levin, S. (1987). Towards a general theory of adaptive walks on rugged landscapes. *Journal of theoretical Biology*, 128(1):11–45.

Khanin, R. and Wit, E. (2006). How scale-free are biological networks. *Journal of computational biology*, 13(3):810–818.

Lima-Mendez, G. and Van Helden, J. (2009). The powerful law of the power law and other myths in network biology. *Molecular BioSystems*, 5(12):1482–1493.

Lin, S., Ji, R., Li, Y., Deng, C., and Li, X. (2019). Toward compact convnets via structure-sparsity regularized filter pruning. *IEEE transactions on neural networks and learning systems*, 31(2):574–588.

Liu, Z., Li, J., Shen, Z., Huang, G., Yan, S., and Zhang, C. (2017). Learning efficient convolutional networks through network slimming. In *Proceedings of the IEEE international conference on computer vision*, pages 2736–2744.

Mehra, P. and Hintze, A. (2022). Evolutionary dynamics in the nk treadmill fitness landscape. *Available at SSRN 4209350*.

Miller, B. L., Goldberg, D. E., et al. (1995). Genetic algorithms, tournament selection, and the effects of noise. *Complex systems*, 9(3):193–212.

Murtagh, F. (1991). Multilayer perceptrons for classification and regression. *Neurocomputing*, 2(5-6):183–197.

Otto, S. B., Rall, B. C., and Brose, U. (2007). Allometric degree distributions facilitate food-web stability. *Nature*, 450(7173):1226–1229.

Pitzer, E. and Affenzeller, M. (2012). A comprehensive survey on fitness landscape analysis. *Recent advances in intelligent engineering systems*, pages 161–191.

Quayle, A. P. and Bullock, S. (2006). Modelling the evolution of genetic regulatory networks. *Journal of theoretical biology*, 238(4):737–753.

Ravasz, E. and Barabási, A.-L. (2003). Hierarchical organization in complex networks. *Physical review E*, 67(2):026112.

Stanley, K. O., Clune, J., Lehman, J., and Miikkulainen, R. (2019). Designing neural networks through neuroevolution. *Nature Machine Intelligence*, 1(1):24–35.

Wagner, A. (2003). How the global structure of protein interaction networks evolves. *Proceedings of the Royal Society of London. Series B: Biological Sciences*, 270(1514):457–466.

Wolpert, D. H. and Macready, W. G. (1997). No free lunch theorems for optimization. *IEEE transactions on evolutionary computation*, 1(1):67–82.