OPTIMIZING FOR MENTAL REPRESENTATIONS IN THE EVOLUTION OF ARTIFICIAL COGNITIVE SYSTEMS

By

Douglas Andrew Kirkpatrick

A DISSERTATION

Submitted to
Michigan State University
in partial fulfillment of the requirements
for the degree of

Computer Science—Doctor of Philosophy

2021

ABSTRACT

OPTIMIZING FOR MENTAL REPRESENTATIONS IN THE EVOLUTION OF ARTIFICIAL COGNITIVE SYSTEMS

By

Douglas Andrew Kirkpatrick

Mental representations, or sensor-independent internal models of the environment, are used to interpret the world and make decisions based upon that understanding. For example, a human sees dark clouds in the sky, recalls that often dark clouds mean rain (a mental representation), and consequently decides to wear a raincoat. I seek to identify, understand, and encourage the evolution of these representations *in silico*. Previous work identified an information-theoretic tool, referred to as $\mathcal{R}$, that measures mental representations in artificial cognitive systems (e.g., Markov Brains or Recurrent Neural Networks). Further work found that selecting for $\mathcal{R}$, along with task performance, in the evolution of artificial cognitive systems leads to better overall performance on a given task. Here I explore the implications and opportunities of this modified selection process, referred to as $\mathcal{R}$-augmentation. After an overview of common methods, techniques, and computational substrates in Chapter 1, a series of working chapters experimentally demonstrate the capabilities and possibilities of $\mathcal{R}$-augmentation. First, in Chapter 2, I address concerns regarding potential limitations of $\mathcal{R}$-augmentation. This includes an refutation of suspected negative impacts on the system's ability to generalize within-domain and the system's robustness to sensor noise. To the contrary, the systems evolved with $\mathcal{R}$-augmentation tend to perform better than those evolved without, in the context of noisy environments and different computational components. In Chapter 3 I examine how $\mathcal{R}$-augmentation works across different cognitive structures, focusing on the evolution of genetic programming related structures and the effect that augmentation has on the distribution of their representations. For Chapter 4, in the context of the all-component Markov Brain (referred to as a Buffet Brain, see [Hintze et al., 2019]) I analyze potential reasons that explain why $\mathcal{R}$-augmentation works; the mechanism seems to be

based on evolutionary dynamics as opposed to structural or component differences. Next, I demonstrate a novel usage of $\mathcal{R}$-augmentation in Chapter 5; with $\mathcal{R}$-augmentation, one can use far fewer training examples during evolution and the resulting systems still perform approximately as well as those that were trained on the full set of examples. This advantage in increased performance at low sample size is found in some examples of in-domain and out-domain generalization, with the "worst-case" scenario being that the networks created by $\mathcal{R}$-augmentation perform as well as their unaugmented equivalents. Lastly, in Chapter 6 I move beyond $\mathcal{R}$-augmentation to explore using other neuro-correlates - particularly the distribution of representations, called smearedness - as part of the fitness function. I investigate the possibility of using MAP-Elites to identify an optimal value of smearedness for augmentation or for use as an optimization method in its own right. Taken together, these investigations demonstrate both the capabilities and limitations of $\mathcal{R}$-augmentation, and open up pathways for future research.

Copyright by
DOUGLAS ANDREW KIRKPATRICK
2021

ACKNOWLEDGMENTS

First and foremost, I would like to thank my advisors, Arend Hintze and Chris Adami, who have guided and mentored me through these 6 long years. Without their advice and wisdom, this dissertation would not have come to fruition.

I would also like to thank my fiancée, Amber Goguen, along with our cat, Rufus, and our dog, Badger. Without your constant love and support, I would not have made it this far.

My parents, Brian and Jackie Kirkpatrick, and my siblings, Brennan and Moe, have also been a constant source of encouragement and support. Thank you; I made it, even if it was a little late (perhaps as expected).

Although the pandemic has hindered some of the lively banter and discussion that provided a wealth of inspiration for this work, I would like to thank the members of the Hintze and Adami labs, past and present, for their support. In particular, I send my thanks to Cliff Bohm, Jory Schossau, Vincent Ragusa, Acacia Ackles, Ali Tehrani, C.G. Nitash, and Leigh Sheneman for their camaraderie over the years.

I would like to thank my other committee members, Charles Ofria and Bill Punch, for their guidance and patience.

I would like to thank the other professors and instructors who have given me advice and guidance over the years, in particular Josh Nahum, Richard Enbody, and Kevin Ohl.

I would like to thank Michigan State University for supporting me over the past 6 years. Without the wonderful resources available through ICER, the HPCC, and BEACON, I would never have made it this far.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

Introduction

True generalized intelligence, as far as we know, has only ever arisen by evolving through biological processes. In recent history, a number of attempts have come to the fore with the aim of producing an approximation of or an equivalent to general or human-level intelligence. These attempts include various approaches in machine learning and deep learning, which generally tune a particular system or network to best learn a set of training data and subsequently also perform well on testing data. These systems produce impressive results, particularly in the game playing (Go [Fu, 2016], recently) and object recognition (e.g., on the MNIST [Deng, 2012, Mohapatra et al., 2015] or CIFAR [Krizhevsky et al., 2009, Liu and Deng, 2015] datasets) domains. These systems gain mastery of specific tasks, but often seem to lack the ability to generalize in broader directions. With the goal of overcoming these issues and creating a general artificial intelligence, I study neuro-evolution, or the use of genetic algorithms to generate artificial cognitive systems or neural networks. By approaching the problem from a bio-inspired perspective, neuro-evolution can follow in the footsteps of biological intelligence and hopefully produce a true generalizable human-level artificial intelligence.

The genetic algorithms that are used for optimizing artificial cognitive systems require an assessment of how well different systems perform relative to each other, either in terms of

specific task performance (e.g., the number of decisions made correctly in a game) or by some other measure. Evolving cognitive systems solely using task performance may ultimately lead to the rise of intelligent behavior but this may take a longer period of time than is feasible for experimental or practical purposes. To combat this defect, we need to examine alternate ways of achieving better task performance in shorter evolutionary time. Thankfully, as we no longer need to strictly follow the whims of biological evolution, we are at liberty to tune the process to our liking. That is not to say that we should disregard the operation of biological evolution in its entirety; to the contrary, we want to take the best parts of biological evolution and tune them even further.

Although task performance may be sufficient to define a fitness function and a genetic algorithm, such a simple function lacks the nuance and complexity of biological evolution. Biological organisms are often evaluated on a number of factors at once. For example, a mouse's fitness depends on its ability to avoid predators, find food, and interact with other mice. Although not directly measured, the mouse's ability to understand its environment is still a strong component of its evolutionary fitness. A mouse that can, for example, construct a mental model of what the warning signs of predators are, or what the shape and smell of a good food source is, will be more likely to survive and have offspring. Thus mental models and an ability to understand one's environment may have a real impact on fitness in biological evolution.

Fortunately, *in silico* we can subvert the intermediary steps of building up a repertoire of tasks and directly attempt to measure and select for the agent's *understanding*, or mental models. If we can successfully identify useful mental models that an agent produces, and select for those along with task performance, we can potentially boost task performance overall by creating an opportunity for exaptation. The model would be selected for earlier in evolution as part of the modified fitness function, so that it can later turn into a useful part of the network and complete the desired task. Such an approach would be analogous to informing someone of a specific sequence that will occur in a video game, and enforcing

that they remember how to exploit it. Although that memory may not improve their ability to beat the game at the exact moment, by ensuring that they remember how to exploit the sequence, it improves their task performance (e.g., their ability to complete the game) later on.

As we have complete control over these artificial cognitive systems, we can access internal states to analyze what they know about their actions. To borrow terminology from philosophy and psychology, we can examine their internal representations - information that they have integrated from the environment into a usable form. To establish a common vernacular for these investigations, the following sections describe a variety of computational substrates, methods, genetic algorithms, and tasks used over the course of this dissertation.

1.1 Computational Substrates for Artificial Cognitive Systems

This dissertation, in order to pursue an understanding of memory and mental representations that is broadly applicable, utilizes a variety of cognitive systems and structures. These structures vary in their origin and functionality, but all share a common thread in that they have a recurrent memory component. This memory structure is vital, as without this structure, representations as I define them are impossible. Each system also has the ability to read from a set of inputs and write to a set of outputs, which allows the cognitive system to interact with the task environments. Lastly, each cognitive system performs a series of computational operations to transform the data from the inputs to the outputs. The cognitive systems outlined below include Markov Brains, Artifical or Recurrent Neural Networks, Genetic Programming Trees, NeuroEvolution of Augmenting Topology components, and Long Short-Term Memory networks. Markov Brains [Hintze et al., 2017] have composable computational components with a flexible, sparse network. Artifical Neural Networks (ANNs) [Russell et al., 2003] are a broad class of networks where a matrix of weights determines the calculation of values from

one layer to the next, working from inputs to internal layer(s) to outputs. I use a subset of this class, the Recurrent Neural Network (RNN), that has an additional recurrent state added to both the input and output layers. Long Short-Term Memory Networks (LSTMs) [Hochreiter and Schmidhuber, 1997] are a different subset of ANNs, where the internal layers and computational processes are strictly defined to provide both long-term storage of information and short-term memory. NeuroEvolution of Augmenting Topologies (NEAT) defines a algorithm for generating ANNs or RNNs. Genetic Programming (GP) trees are defined as a tree-like structure of mathematical operations [Koza, 1994, Miller, 2011], where the inputs to the operations are either constants, input or recurrent node values, or the results of earlier computations in the tree. Although each of these systems follows the same broad structure of input to computational process to output, it is in the differences both major and minor that I can examine the true nature of representations.

1.1.1 Markov Brains

Markov Brains are the primary cognitive system used in the research for this dissertation. A Markov Brain, in the abstract form, is a series of input, output, and hidden (i.e., recurrent) nodes, which are read from and written to by a series of computational modules or gates. As a result, Markov Brains are sparse - their connectivity must be evolved - and highly composable. Each gate can be defined as any input/output pattern matching system or network, up to and potentially including other cognitive systems, including for example another Markov Brain. However, in this dissertation I limit the gates available to evolution to a subset of all possible gates. The canonical version of Markov Brains uses deterministic and probabilistic logic gates with between 1 and 4 inputs and between 1 and 4 outputs. A more recent version of the Markov Brain, referred to as a "Buffet Brain" [Hintze et al., 2019], uses those two original gates in addition to ANN gates, GP gates, and NEAT gates. Each gate reads from (i.e., is wired from) a set number of input or hidden nodes or states and writes to (i.e., is wired to) a set number of output or hidden nodes. The number of input and output states

are task-dependent, while the number of hidden or recurrent nodes is defined by the user (typically, 8 hidden nodes are used in this work). The brain structure is defined by a genome, where certain values on the genome indicate a coding region for a particular gate type, and the subsequent values define the functionality of that specific gate (e.g., its input wiring, output wiring, and internal logic table in the case of a deterministic logic gate). Genomes are seeded with a set number of gates of the types specified for use at the start of each evolutionary run. The Markov Brain structure is intended to be highly modular and sparsely connected.

1.1.2 ANNs and RNNs

The prototypical ANN as used in this work is a feed-forward perceptron network [McCulloch and Pitts, 1943, Minsky and Papert, 1969], with a hyperbolic tangent activation function. A single-layer version of the ANN is used as a gate type in Markov Brains, while RNNs are used as a standalone brain type. ANNs and RNNs as used here are both fully connected, with the values of each weight determined by a matrix read from the genome. RNNs differ from the ANN in that they have additional recurrent nodes that can be read from into the first layer along with inputs, and written to the last layer along with the outputs. The values of recurrent nodes persist across updates, allowing for persistent memory. Although many versions of ANNs and RNNs exist, with the notable exception of LSTMs the acronyms as used in this dissertation refer to the simple, straightforward versions described here. Although I have no reason to doubt the applicability of the methods described here to other network types, for the sake of efficiency I will focus only on the fully-connected ANN and RNN, as opposed to transformers, convolutional neural networks, or other recent variations.

1.1.3 GP Trees, Gates, and Networks

Genetic programming (GP) in this dissertation refers primarily to GP trees, where each node in the tree is either a function, a constant value, or an input value. The mathematical

operations or functions take the values coming from one or more of its children nodes to calculate a new value that is propagated up the tree. The node functions can take the form of any mathematical or logical function. In the GP gate used as part of the Markov Brain, I use a more limited subset of common unary and binary input functions (see [Hintze et al., 2019] for a complete list) to provide computational capability. GP Trees have been used to evolve functions that model real-world process, and have also been adapted to create agent controllers. For the full network structures used here, particularly in Chapter 3, I look at the versions that serve as agent controllers. I use two primary types of GP Tree based networks, the GP-Forest and GP-Vector Brains. For the GP-Forest Brain, I construct an indirectly-encoded GP Tree for each output and hidden/recurrent node. Each node in the tree is a function, a constant, or the previous value of an input or hidden node. For the GP-Vector Brain, vector operations are used instead of unary or binary operations on individual values. Each node is either a constant vector, the entire state vector (input, output, and recurrent/hidden states combined), or the result of a mathematical operation applied to all elements in the child node vector value(s).

1.1.4 NEAT Gates

While the ANN and GP gates are described in conjunction with the corresponding standalone brain type descriptions, here I describe the NEAT gate used by the Markov Brain in a few parts of this dissertation. The NEAT gate type, described comprehensively in [Hintze et al., 2019], is a hybrid between the GP and ANN gates. As used here, it takes a set of inputs, with a weighted summation and hyperbolic tangent activation function (similar to the ANN gate) and applies a mathematical function to the result, similar to the GP gate. This formulation of the NEAT gate draws on the mathematical transformation of networks concept sourced from the original NEAT work.

1.1.5 LSTM Networks

The Long Short-Term Memory network (LSTM) is a variation of the RNN specifically designed so that there are two main channels of memory: the cell states and the hidden states. The mathematical operations that describe the connectivity of each of these layers are designed to maintain one set of states as a short-term memory analogue (the hidden states), while the other is meant to track more closely with biological long-term memory storage (the cell states). Both the cell states and hidden states are recurrent, and as such the total number of hidden and cell states is set in this work to equal the number of recurrent states in the other cognitive system types, expanded as needed by padding the cell layer. A number of sigmoid and hyperbolic tangent functions operate directly on the hidden nodes and integrate information directly from the inputs, causing it to be more volatile. The opportunity to manipulate the cell state, on the other hand, is gated by sigmoid and point multiplication operations, leading to higher long-term stability.

1.2 Mental Representations

Mental representations have a long and storied history in philosophy, psychology, and cognitive science. The idea that the mind has some sort of internal understanding has been a central point across multiple disciplines, and underlies our understanding of biological intelligence. To be clear, I refer here to mental representations as internal, sensor-independent models of the world possessed by an individual. These representations can range from the simplest of concepts, including amounts or relative values, all the way up to complex understandings of other individuals and the actions they might take [Collins and Gentner, 1987]. Although it is clear that these representations are present in biological and human instances, their utility in the creation of artificial cognitive systems has been questioned. Some have gone so far as advocating to abandon representations altogether, and to instead pursue "intelligence without representation." [Brooks, 1991] This work emphatically disputes that notion.

'Intelligent' systems without representations would be purely reactive systems, unable to respond to dynamic environments when prior states or information must be considered to make the optimal decision. Instead I posit that representations are a necessary component of artificial cognitive systems, and furthermore that the creation of such representations must be encouraged to create truly intelligent artificial cognitive systems. Previous work supports this notion, both that mental representations exist in artificial cognitive systems [Marstaller et al., 2013], and also that by encouraging mental representations I may create systems that perform better on a given task [Schossau et al., 2015]. The crux of this work is to better understand how such representations form, and how I may better encourage the evolution of representation and better-performing artificial cognitive systems.

1.3 Neurocorrelates and Information Theory

In order to assess mental representations in artificial cognitive systems, I must quantitatively evaluate their presence or absence. To do so, I turn to neurocorrelates, which are measured correlations between the network state and conditions related to the environment, either in what the actual relevant state of the environment is or what information the agent is presented with. I use information-theoretic measures to formally quantify these neurocorrelates.

1.3.1 Entropy Measurement and Pseudo-counts

In order to calculate a Shannon entropy, one can either find the mathematical real value or come to an approximation of that value from sampling. Here I use the latter, finding the shared information by an approximate calculation over the observed network behavior, generated by running the cognitive systems through the environment and recording relevant states. I can then apply Shannon entropy calculations to calculate correlations, entropy, and shared information between relevant world (or environmental), recurrent (or hidden), and sensor (or input) states. That is, I record the state of the world, the hidden states of the

cognitive system, and the states that the system is receiving at every time point during the execution of the task. The hidden states of the brain collectively form the independent variable B, the environmental states form the independent variable E, and sensor states the independent variable S. The probabilities of each state are determined by counting the number of times each state occurs over the course of the agent's lifetime. These counts conducted over the lifetime of the agents are only a portion of the state space, and thus are a pseudo-count or approximation of the true counts. For example, if the agent's lifetime was 10 time steps long, and a certain sensor state s_q appeared twice, the probability $P(s_q)$ would be 0.2. The probability for each state of each variable and joint combinations of different variables are calculated, and used to calculate the Shannon entropies. The Shannon entropy H for a given variable X with unique states x_i is shown in Equation 1.1.

$$H(X) = \sum_{i=1}^{n} -P(x_i) \times \log P(x_i) \tag{1.1}$$

A joint entropy for a given set of states (e.g., S and B) would use the probability that the same brain and sensor states occurred simultaneously, also calculated using the pseudo-count method. By calculating the individual entropy of each set of states or the joint entropy of two or three sets of states, I can determine the mutual or shared information between those states. With this shared information, I can formalize our assessment of what representations the cognitive system has about its environment.

1.3.2 $\mathcal{R}$: A measure of Representation

$\mathcal{R}$ is an information-theoretic measure of mental representations in artificial cognitive systems. Formally, it is the information shared between the hidden states of the cognitive system and the environment not currently present in the sensors (i.e., information that has already been integrated into the system, but not what is actively presented). However, this definition does not say that the $\mathcal{R}$ is independent of the sensors entirely; the internal models used can and

often are conditional on previous inputs. This can be explicitly formulated as $I(B : E|S)$, that is the information shared between the entirety of the brain hidden or recurrent states and entirety of the environmental or task states, given all of the information currently present in the sensors. The shared conditional information $I(B : E|S)$ can also be visualized using the information-theoretic Venn diagram, shown in Figure 2.3. Previously, $\mathcal{R}$ was calculated using the expanded form of Equation 3.2, although a portion of this work (primarily 3.3.1) demonstrates how the calculation may be simplified to the more efficient Equation 3.3.

1.3.3 $\mathcal{S}$: A measure of Smearedness

While $\mathcal{R}$ measures the absolute amount of information that the cognitive system stores about the environment, smearedness presents a more nuanced view of *how* the information that the system integrates is spread across the hidden nodes. Formally, smearedness measures how much information about discrete concepts in the environment are spread across, or concentrated in, individual hidden nodes in the cognitive systems. Smearedness was originally introduced in [Hintze et al., 2018], and is used here as a complement to $\mathcal{R}$. In [Hintze et al., 2018], we define two different formulations of smearedness: smearedness of nodes $\mathcal{S}_n$ and smearedness of concepts $\mathcal{S}_c$. To calculate either, I must first quantify the amount of information that each hidden or recurrent node in the cognitive has about each concept of the task or environment. This is done for all nodes n and all concepts c by calculating the 'atomic $\mathcal{R}$', for each pair and filling out the matrix of combinations M as follows. The whole memory state in the joint random variable B are separated into the individual random variables B_i, where $B = B_1B_2...B_y$. Likewise, the environmental state variable E is decomposed into the individual random variables E_j, where $E = E_1E_2...E_z$ Each individual entry in the matrix $M_{i,j}$ is defined for a memory node B_i and a concept E_j, and then is calculated in a manner identical to $\mathcal{R}$, but only considering the individual node and concept instead of the entire state space for each. That is, each matrix element of M is calculated as $M_{i,j} = I(B_i : E_j|S)$. Using M, I can take a pairwise minimum either over each node for $\mathcal{S}_n$ or over each concept

for $\mathcal{S}_c$ and summed to create the final calculation. The formula for $\mathcal{S}_c$ is shown in Equation 1.2, and the formula for $\mathcal{S}_n$ is shown in Equation 1.3.

$$\mathcal{S}_c = \sum_i \sum_{j>k} min(M_{ji}, M_{ki}) \tag{1.2}$$

$$\mathcal{S}_n = \sum_i \sum_{j>k} min(M_{ij}, M_{ik}) \tag{1.3}$$

In recent work [Kirkpatrick and Hintze, 2019] we show that the two formulations of smearedness, $\mathcal{S}_c$ and $\mathcal{S}_n$ broadly correlate, and here I will use the notation $\mathcal{S}$ to refer to the smearedness of nodes, $\mathcal{S}_n$, or to the broader concept of smearedness.

1.4 Optimization Methods and Genetic Algorithms

In order to create a network that controls an agent with the goal of completing a given task, I must be able to refine and optimize the network for better performance. There are many algorithms used for these optimizations [LeCun et al., 1998,Kaur et al., 2018,Yao, 1999,Stanley and Miikkulainen, 2002], although for this work I focus on the a subset of optimization methods that fall under the category of Evolutionary Optimization, and more specifically Genetic Algorithms (GAs). GAs operate primarily on systems that can be described using a genome, and that are typically organized in populations of different networks. Here I use circular genomes with point, insertion, and deletion mutations in populations of typical size 100. In most GAs, the population or a portion of the population is evaluated on the task using a fitness function, and the best performers are selected to create the next generation. I use two primary GAs in this work, Roulette Selection and MAP-Elites, and investigate how using $\mathcal{R}$ and $\mathcal{S}$ in conjunction with fitness in the GAs can improve the task performance of agents produced by the GA.

1.4.1 Roulette Selection

The primary GA used in this work is Roulette selection, or fitness-proportionate selection. Suppose agents are evaluated on a task, resulting in a fitness ω. In fitness-proportional selection, the agents are chosen to propagate to the next generation with probability proportional to ω. For example, if there are two agents in a population where one has fitness $\omega = 7$, and the other $\omega = 3$, the first has a 70% chance of being selected to be in the next generation, while the other has a 30% chance of being selected for the next generation. Under this selection regime, the proportion of different fitness values matters, and so multiplicative increases in fitness that happen under $\mathcal{R}$-augmentation may have a larger effect than under other selection regimes like Tournament selection where the relative sizes would not matter, but only whether one fitness was greater than the other.

1.4.2 MAP-Elites

MAP-Elites is a genetic algorithm designed to maintain phenotypic or genotypic diversity among the organisms in the population. To achieve this goal, different measures are established as axes in a multidimensional space, and further subdivided into subsections by selecting different ranges of values for those measures. The intersection of these subsections creates bins in which organisms meeting the criteria for falling into that bin (i.e., by having the values for each measure or metric fall into a specific range) only compete with each other on a final metric, typically task performance. The MAP-Elites algorithm can be described as follows. I create an initial population, and evaluate each agent on the different metrics. I classify each organism or agent in the population into the correct bin based on the values of its corresponding metrics. If two or more agents fall into the same bin, I maintain the agent (or agents in some variations) with the highest fitness, and discard the others. I then select organisms from the population stored in the grid at random, evaluate them on the various metrics, and assign the agents to bins as specified. Again, if any organisms fall into the same

bin as a preexisting organism, only the best is kept. I show this process in Figure 1.1 This allows for the exploration of the genotypic or phenotpyic space, as well as identifying the values of the metrics which allow for optimal performance. As used in this work, the axes for MAP-Elites are defined by $\mathcal{R}$ and $\mathcal{S}$, and task performance (ω) is used as the deciding metric.

1.4.3 Multi-objective optimization and $\mathcal{R}$-Augmentation

Many different options exist for multi-objective optimization [Deb, 2001], ranging from particle swarm optimization [Parsopoulos and Vrahatis, 2002] to directed search [Wang et al., 2017] or other various explorations of the Pareto-front [Konak et al., 2006]. Although I view these methods as viable paths of exploration for finding agents that achieve higher task performance and have better internal representations, I do not seek to compare them in the context of this work. The integration of such methods would introduce additional complexity and distract from our goal of understanding how representations are formed and can be used to create agents with higher task performance. Here I narrow my focus on a simple multi-objective method that has been shown to accomplish the stated goals [Schossau et al., 2015]. This method, referred to as $\mathcal{R}$-augmentation, involves modifying the original fitness function (e.g., Equation 2.2) so that the measured mental representation, $\mathcal{R}$, is also taken into account and increases in $\mathcal{R}$ are rewarded alongside fitness. This results in the modified fitness function, shown in Equation 2.3. My primary focus in this thesis is on demonstrating the applicability and utility of $\mathcal{R}$-augmentation, as well as investigating what factors allow the method to produce agents with better task performance. I also explore some alternative methods of augmentation and multi-objective optimization by the inclusion of alternate neurocorrelates (namely $\mathcal{S}$), including the use of the MAP-Elites algorithm, in Chapter 6.

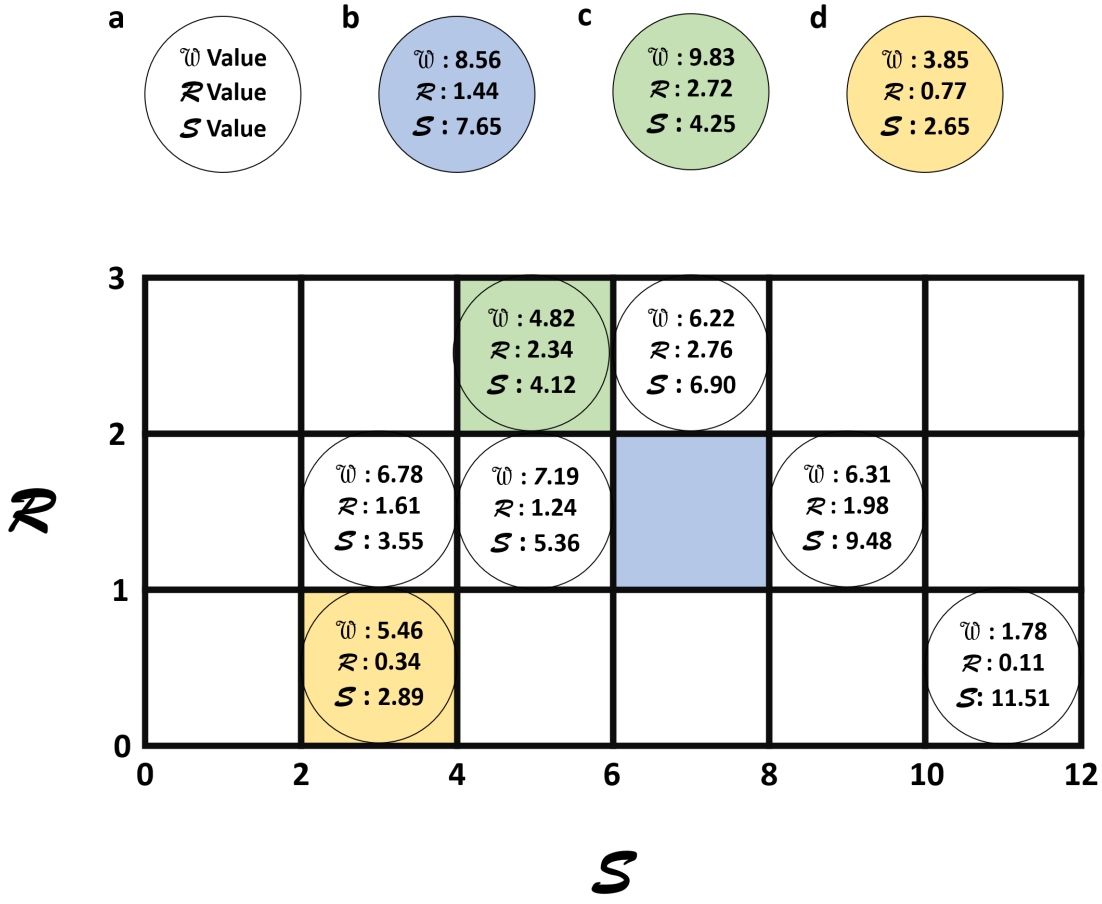

Figure 1.1: A simplified version of the MAP-Elites algorithm. Each organism is represented by the circle, and each square represents a bin for the algorithm. A prototypical organism is labeled (a), and demonstrates that each agent has a task performance score ω along with a measurement of the agent's $\mathcal{R}$ and $\mathcal{S}$. The grid is defined in terms of $\mathcal{S}$ on the x-axis and $\mathcal{R}$ on the y-axis. The bins for $\mathcal{R}$ are from 0 to 1, 1 to 2, and 2 to 3 while the bins for $\mathcal{S}$ are from 0 to 2, 2 to 4, 4 to 6, 6 to 8, 8 to 10, and 10 to 12. The algorithm will place the organism labelled (b) in the square with the same light blue shade based on the organism's $\mathcal{R}$ and $\mathcal{S}$ values. The organism will be successfully placed in the bin because there are no organisms in that bin previously. The algorithm will place the organism labelled (c) in the square with the same light green shade based on the organism's $\mathcal{R}$ and $\mathcal{S}$ values. The placement will succeed as the organism labelled (c) has a higher task performance than the organism already in the cell. The algorithm will attempt to place the organism labelled (d) in the bin of the same light yellow color based on the organism's $\mathcal{R}$ and $\mathcal{S}$ values. The placement will fail, however, because the organism that is already in the bin will remain because it has a higher task score, and the organism labelled (d) will be discarded.

1.4.4 Machine Learning

While GAs are the only method of optimization addressed in this work, it should be noted that the information-theoretic methods used here should be no less applicable to networks or systems optimized using Machine Learning (ML) techniques such as gradient descent for backpropagation. Given the unpredictable and stochastic nature of evolutionary optimization, the performance benefits of $\mathcal{R}$-augmentation are especially notable. That being said, adding $\mathcal{R}$ as a portion of metrics calculated for the loss function in ML applications has the potential to produce similar gains in task performance. The increased size of ML-trained networks, particularly in the context of Deep Learning, may limit the applicability of $\mathcal{R}$ as a training metric unless further optimizations are made. Regardless, $\mathcal{R}$, $\mathcal{S}$, and other information-theoretic tools may yet prove to be useful in the analysis of black-box systems produced by ML methods.

1.5 Tasks

To evolve intelligence, I must first have a world or environment for intelligent agents to evolve in. Moreover, when seeking to evolve or assess mental representations on these cognitive systems, I must have a well-defined environmental state to serve as a starting point for the evaluation of the information-theoretic measures used here. Tasks that lack a well-defined environmental state (i.e., a state that is useful to the cognitive system) will not produce meaningful results in terms of the information-theoretic neurocorrelates. Rather than create new tasks to serve our purposes, in this dissertation I use two tasks identified in previous work as having suitable environmental states: the Active Categorical Perception Task [Marstaller et al., 2013], and the Number Discrimination Task [Kirkpatrick and Hintze, 2019].

1.5.1 Active Categorical Perception Task

The Active Categorical Perception Task (ACP) is a primary task used in the study of representations in evolved artificial cognitive systems [Marstaller et al., 2013, Schossau et al., 2015, Hintze et al., 2018]. The task itself, which originates in work on perception [Beer, 1996, Beer, 2003, van Dartel et al., 2005], is relatively straightforward; an agent sees a block falling towards it, and must either catch or avoid it based on an observed property or set of properties of the block. The original version of the task [Marstaller et al., 2013] has an agent with a pair of sensors of width 2 facing upwards (i.e., towards the falling blocks), separated by a break of width 2 between the sensor blocks (i.e., a blind spot). Blocks fall downwards and either to the left or the right at a constant pace. In the default task, small blocks (of size 2) must be caught while large blocks (of size 4) must be avoided, but these requirements may be changed to suit the experimenter. A block is deemed to be caught if any part of the agent (sensor or blind spot) makes physical contact with the block. The agent can move to the left or to the right, or remain still on any given time step. The world is toroidal (i.e., if the agent or block goes too far to the left it will wrap around to the right, and vice versa) with a width of 16 and a height of 32. For each evaluation of the agent, every possible permutation of block size (large or small), direction (falling left or right), and starting position relative to the agent (out of the 16 positions in the environment) is used to test the agent. The environmental states that are used for analysis in $\mathcal{R}$ and $\mathcal{S}$ are whether the block is large or small, whether the block is falling to the left or to the right, and whether the block is to the left or to the right of the agent at the current time step. Although other variants of the task are possible (e.g., using blocks that change shape as they fall), for this work I will keep the original layout of the task as described here unless explicitly stated otherwise. The agent is evaluated on the number of correct (C) and incorrect (I) catch or avoid decisions made over all permutations.

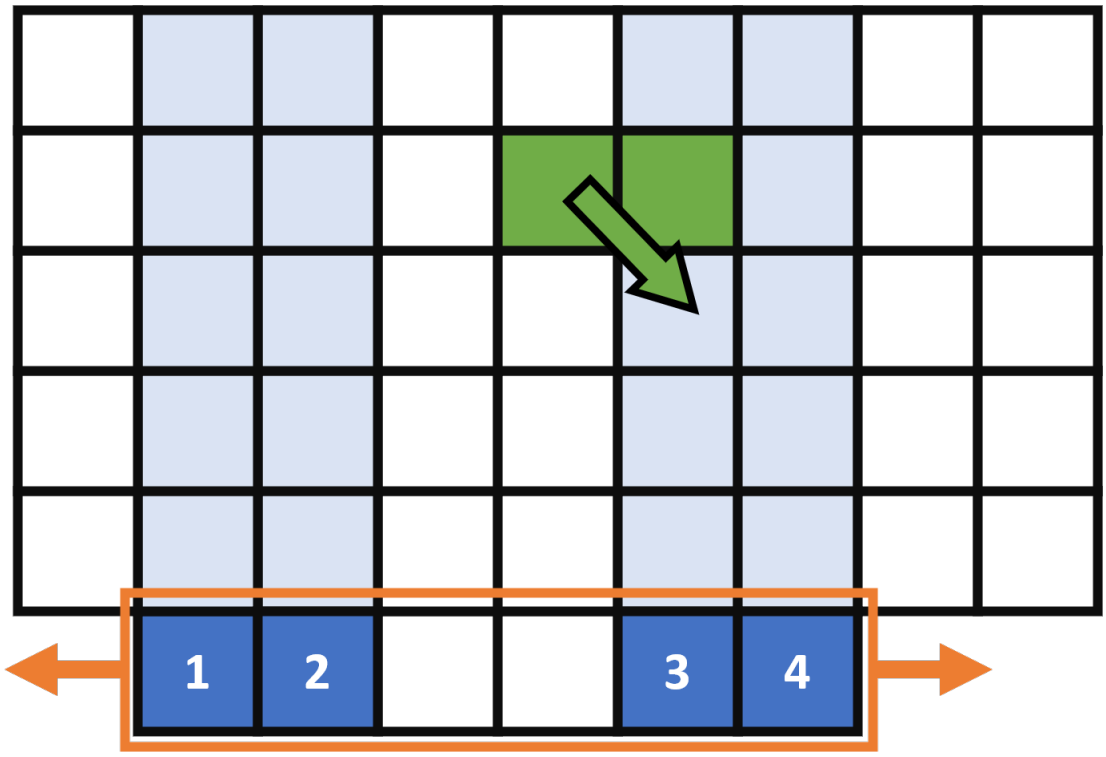

Figure 1.2: A subsection of a single case of the Active Categorical Perception Task. The agent, visualized by the orange box, can move either to the left or right as indicated by the orange arrows. The four sensors, labelled 1,2,3, and 4, can each see directly upwards as indicated by the light blue shading. A small block with width 2 is falling towards the agent and moving to the left, as indicated by the green arrow. At the current point in time, sensor 3 is the only sensor that would return a signal, while on the next update, assuming that the agent did not move, both sensors 3 and 4 would return a signal.

1.5.2 Number Discrimination Task

The Number Discrimination Task is a secondary task used for the evaluation of representations in evolved artificial cognitive systems, adopted more recently [Kirkpatrick and Hintze, 2019] *in silico*. The task, however, has its roots in biological [Nieder, 2018, Merritt et al., 2009] and psychological [Merritt and Brannon, 2013] analogues. For the task as used here, the agent has a set number of binary inputs N. On the first update, a subset of the inputs x (where $0 <= x <= N$) are set to 1, and the remainder are set to 0. After a given number of intermediate updates i, where the agent may receive a constant signal or noise, the agent then receives a second signal where a second number of the agent's inputs y are set (where $0 < y <= N$ and $y \neq x$). After a second set of intermediate updates, again of length i, the agent is given a final constant signal where it must output a 1 if the $x > y$, and a 0 otherwise. The agent will see all possible permutations of x and y, both in the variations of patterns shown (i.e., for $N = 5$, $x = 2$ could present as 00101 or 11000) and variations in the pairs of numbers (i.e., for $N = 5$, some examples would be $x = 2$ and $y = 3$, $x = 3$ and $y = 2$, $x = 5$ and $y = 1$, or $x = 4$ and $y = 5$). I show an example of one case of the task in Figure 1.3. The relevant environmental states used for $\mathcal{R}$ and $\mathcal{S}$ are a binary state corresponding to which values of x and y are currently being tested, and a binary state indicating if the first number was larger than the second. The agents are evaluated on the number of correct (C) and incorrect (I) evaluations of the ordering over all permutations.

1.6 Overview

This work focuses on the use and effects of augmenting the fitness function to emphasize mental representations in artificial cognitive systems. In Chapter 2 I study the effect of $\mathcal{R}$-augmentation on different computational components, and investigate at the cognitive system's ability to generalize to within-domain tasks and ability to resist sensor noise, while the performance advantages are broadly retained. In Chapter 3 I look at the effect of augmentation

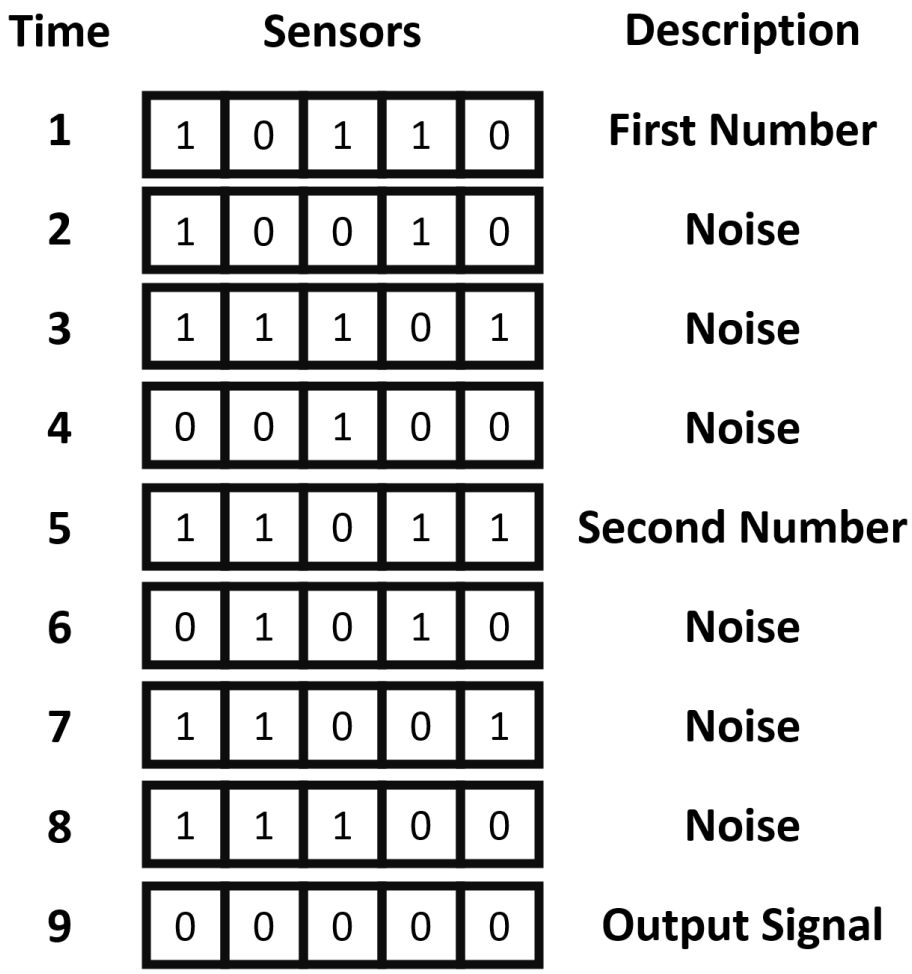

Figure 1.3: A single case of the Number Discrimination Task. The agent receives a series of inputs over a period of 9 brain updates. The input values are shown in the squares under "Sensors", where each square represents a unique sensor input, and correlate with brain updates listed under "Time". The description of each sensor input is likewise listed under "Description". The agent receives a first number, here 3 (note that 3 of the sensors have a 1 as input), and then has 3 intermediate updates where noise is fed to each sensor with probability $p = 0.5$. The agent then receives a second number, here 4 (note likewise that 4 of the inputs are 1), and then a second period of 3 intermediate updates where noise is fed to the sensors with the same probability p. After the second set of noise inputs, the agent receives an output signal of all '0' inputs, and is expected to output a 1 because the second number is larger than the first.

on different network structures. The advantage in performance seen primarily in Markov Brains is also seen in GP-related structures and ANNs. Chapter 4 explores how augmentation affects brains made of multiple computational components in the same structure, and use this to examine reasons as to why augmentation enhances the evolutionary regime. In Chapter 5 I focus on the idea that $\mathcal{R}$-augmentation can allow cognitive systems to integrate more information from the environment and maintain task performance as sample size decreases (i.e., as fewer examples from the task are shown during evolution). This hypothesis is examined in the context of in-domain and out-domain generalization (i.e., $\mathcal{R}$-augmentation allows better performance on unseen trials, across different evolutionary sample sizes). Lastly, Chapter 6 goes beyond the scope of simple $\mathcal{R}$-augmentation, where I examine how augmentation could target a value beyond simple minimization and maximization. Specifically, I look at augmenting the fitness function to bring smearedness to an optimal value. In order to determine this optimal value, I use MAP-Elites to examine the fitness landscape in terms of respresentation $\mathcal{R}$ and smearedness $\mathcal{S}$, and investigate the potential use of MAP-Elites as an optimizer in its own right.

Chapter 2

Addressing concerns regarding potential limitations or effects of $\mathcal{R}$-augmentation

A portion of this chapter was originally published as "Kirkpatrick, D., & Hintze, A. (2019, July). Augmenting neuro-evolutionary adaptation with representations does not incur a speed accuracy trade-off. In Proceedings of the Genetic and Evolutionary Computation Conference Companion (pp. 177-178)." Significant portions were edited or added for inclusion in this dissertation.

2.1 Abstract

Representations, or sensor-independent internal models of the environment, are important for any type of intelligent agent to interact with an environment. Imbuing an artificially intelligent system with a model of the world it functions in remains a difficult problem. However, using neuro-evolution as the means to optimize such an artificial neural network or other similar artificial cognitive system allows the genetic algorithm to evolve networks

with proper models of the environment. Previous work introduced an information-theoretic measure, $\mathcal{R}$, which quantifies how much information a neural computational architecture (henceforth loosely referred to as a brain) has about its environment. In addition, $\mathcal{R}$ can be used speed up the neuro-evolutionary process by including an evaluation of $\mathcal{R}$ as part of the fitness function. However, it is not known whether that this improved evolutionary adaptation comes at a cost to the brain's ability to generalize or the brain's robustness to noise. It is also not known how this process of using $\mathcal{R}$ as part of the selection scheme (referred to as $\mathcal{R}$-augmentation), will perform when different internal mathematical and logical functions (referred to as computational components) are evolved in the same network structure. In this chapter, I show that neither of these issues are a concern. $\mathcal{R}$-augmentation produces results similar to a standard genetic algorithm when examining robustness and ability to generalize, and the improved performance remains constant across different computational components. Furthermore, I find an improved ability of the brain to evolve in noisy environments when $\mathcal{R}$ is used to augment evolutionary adaptation.

2.2 Introduction

Neuro-evolution has shown promising results in creating more capable and adaptable solutions for problems in artificial intelligence [Yao, 1999, Floreano et al., 2008, Stanley et al., 2019]. One of the downsides to using evolution rather than the more standard backpropagation to "train" a network is that evolutionary search itself is inherently random and as a consequence, an optimal agent may not evolve quickly. Similarly, it is possible that the evolutionary search gets stuck, and results in a less than optimal solution. To combat this defect, many methods have been proposed to accelerate neuro-evolution, for example by using an indirect encoding [Stanley and Miikkulainen, 2002], increasing the diversity in the population [De Jong, 1975], or by protecting new mutations with novelty search [Lehman and Stanley, 2008]. Another method rewards solutions that are more modular [Clune et al., 2013], and indeed this method

is technically the closest to the approach used here. Each of these methods shares in common the influence of a measurable quantity on the agents' evolutionary optimization.

One issue that has already been identified to be specifically troublesome in deep learning is representations. It seems as if neural networks are easily fooled [Nguyen et al., 2015] either because they are over-fitted, or because they simply cannot generalize well. Deep neural networks lack a robust internal model, and appear to rely too heavily on specific irregularities in the data used for training. In image recognition these could be surface irregularities in specific images that are learned, for example [Jo and Bengio, 2018]. The fact that these deep learned models do not form true representations mirrors previous work which suggests that one might even get away with creating intelligence without representations [Brooks, 1991]. However, direct programming or training is not the only way to generate internal models; they can be evolved as well. Neuro-evolution allows systems to create cognitive agents that have meaningful and robust representations about their environment [Marstaller et al., 2013]. Furthermore, it has been shown that one can augment a genetic algorithm (GA) using this measurement of representations to improve network task performance relative to a standard GA [Schossau et al., 2015]. This process works by first measuring representations and then rewarding the agent for having these representations by including the measurement in the fitness function. This is similar to a multiple objective optimization [Zhou et al., 2011], but where both parameters are mostly in line with the same goal.

It might at first sight seem impossible to measure "how much a brain is modelling about the world" but as the purpose of these models is to make predictions about the world, there must be a correlation between the brain and the world that can be measured quantitatively using information theory. Such a measure, referred to as $\mathcal{R}$, was introduced in 2013 and quantifies the information shared between a brain's internal states and the environment that is not shared with the sensors. Colloquially, $\mathcal{R}$ is the information that the brain knows about the environment that is not coming from the sensors. Previous work has shown that using $\mathcal{R}$ as a co-evolutionary factor (i.e., by rewarding the agent for the presence of $\mathcal{R}$ alongside

task performance) increases the rate of adaptation in the genetic algorithm [Schossau et al., 2015]. In the following, I will call this method of using $\mathcal{R}$ as a portion of the fitness function "augmenting the genetic algorithm with $\mathcal{R}$", or $\mathcal{R}$-augmentation.

While the benefits of $\mathcal{R}$-augmentation have been demonstrated on a variety of fitness functions, there is some concern that this method might impair a network's robustness, since it is known that the rate of adaptation and the degree of robustness are tightly connected [Wagner, 2012]. In other words it is possible that augmenting a genetic algorithm with $\mathcal{R}$ might speed up evolution but cause detrimental trade-offs for other aspects of intelligence. Here I investigate two aspects of intelligence that might suffer from trade-offs and are relevant for artificial intelligence applications: the brain's ability to generalize as well as a brain's resilience to noise.

Often in machine learning, (e.g., when training a classifier), the distinction between robustness and generalization is hard to make, and the terms are often used interchangeably. Here, I define generalization as the ability to deal with a new task the system was not explicitly trained on. Robustness on the other hand is the ability to perform the same task even with significant levels of noise added to sensors.

Previous work [Schossau et al., 2015] has focused on the use of a specific architecture (the Markov Brain) which uses a specific computational processing component, the deterministic logic gate. Other work [Hintze et al., 2019, Hintze et al., 2017] has introduced the possibility of using alternative computational units to expand the capability of the Markov Brain system. However, we do not know how (or even if) $\mathcal{R}$-augmentation works when Markov Brains utilize computational components other than Boolean logic.

In this paper, I show that neither the ability of Markov Brains to generalize nor the brain's robustness to noise is reduced when the evolutionary process is augmented by using $\mathcal{R}$. In particular I find that the $\mathcal{R}$-augmentation process creates an increase in performance regardless of the computational component used. Moreover, I found co-evolving with $\mathcal{R}$ results

in better performance when evolving in noisy environments.

2.3 Materials and Methods

2.3.1 Markov Brains

Markov Brains are a special form of evolvable neural networks, also sharing similarities with Cartesian genetic programming [Miller and Thomson, 2000]. Here they are used to control a virtual agent, and are optimized using a genetic algorithm. The optimization is based on the performance of the controlled agent in a test environment. Like a conventional Recurrent Neural Network (RNN) [Russell and Norvig, 2016] Markov Brains have input nodes that receive data from the agent's sensors, can process information over time using recurrent hidden nodes, and have output nodes that are used as actuators allowing the agent to move in the environment. However, instead of having a fixed, layered topology like an RNN, Markov Brains use computational gates that can read from and write into input, hidden, and output nodes. The way the gates connect to nodes, how many, and which types of logic each gate executes are all evolvable properties. Markov Brains have originally been designed using deterministic [Marstaller et al., 2013] or probabilistic logic gates [Edlund et al., 2011], but have since then been extended to include many types of computational elements [Hintze et al., 2019]. A Markov Brain uses a linear vector of bytes as a genome, with circular wraparound as necessary. The genome is read sequentially, and each time a specific sequence of numbers is found (e.g., 42 followed by 213 for a deterministic logic gate) the following numbers are used to specify the functionality, logic, and connections of each gate. This is loosely analogous to a start codon defining the start of a gene, and the following sequence of nucleotides encoding a protein. For a more detailed description of Markov Brains see [Hintze et al., 2017]. All experiments are implemented using the MABE code framework [Bohm et al., 2017]. For the primary experiments about generalization and robustness, I only use the deterministic logic gate. For the examination of gate types, I run experiments where the brain uses an

individual gate type, with brains constructed using each of the Probabilistic Logic Gate, the Genetic Programming (GP) Gate, the Artificial Neural Network (ANN) Gate, and the NeuroEvolution of Augmenting Topologies (NEAT) Gate.

Markov Brains are evolved for either 40,000 (for the generalization and robustness experiments) or 10,000 generations (for the individual gate experiments) in populations of 100 organisms. The start population is initialized with random genomes of length 5000. At each generation the genomes experience gene duplications (with probability $2x10^{-5}$ per site of the parent genome, duplicate the next 128 to 512 contiguous sites), deletions (with probability $2x10^{-5}$ per site of the parent genome, delete the next 128 to 512 contiguous sites), and point mutations (with probability $5x10^{-3}$ per site of the parent genome, randomize the site). The genome size is capped at a maximum of 20,000 sites and a minimum of 2,000 sites.

2.3.2 Active Categorical Perception Task

I selected a task previously used in the quantification of $\mathcal{R}$ [Marstaller et al., 2013] as well as in the original discovery of $\mathcal{R}$-augmentation [Schossau et al., 2015]. This is done to take advantage of the previously examined and well-formed definitions of the environmental concepts of the task, which is essential to the proper assessment of an agent's representation. The task is referred to as the Active Categorical Perception (ACP) task, and originally comes from previous work on agent control schemes [Beer, 1996, Beer, 2003, van Dartel et al., 2005].

In this task, an agent is endowed with a set of upward-facing sensors, as well as a motor that allows the agent to the left and right, as seen in Figure 2.1. Agents exist in a world with a width of 16 units and a height of 34 units, where the left and right sides connect toroidally. The sensors are two groups of two individual sensor inputs, separated by a space in the middle. Previous work had set the distance between the two groups to be two units, although here I allow for a variation of either 2 or 3 unit spaces between the sensor groups, as shown in Figure 2.2. The sensors are capable of determining whether an object, referred

to as a block, is directly above it or not. The blocks are dropped one at a time towards the agent, and move at a constant rate of 1 unit per time step down and 1 unit either to the left or to the right. Agents have a fixed amount of time (here, 34 time steps) to either be under the block and thus catch it, or to not be under the block and thus avoid it. In previous work, the small and large blocks were of size 2 and size 4, respectively. Here, I vary the size of the large and small blocks, where the small blocks can be between 1 and 3 units wide, and the large blocks can be between 4 and 6 units wide. The task for the agent is to catch the small blocks and avoid the large ones.

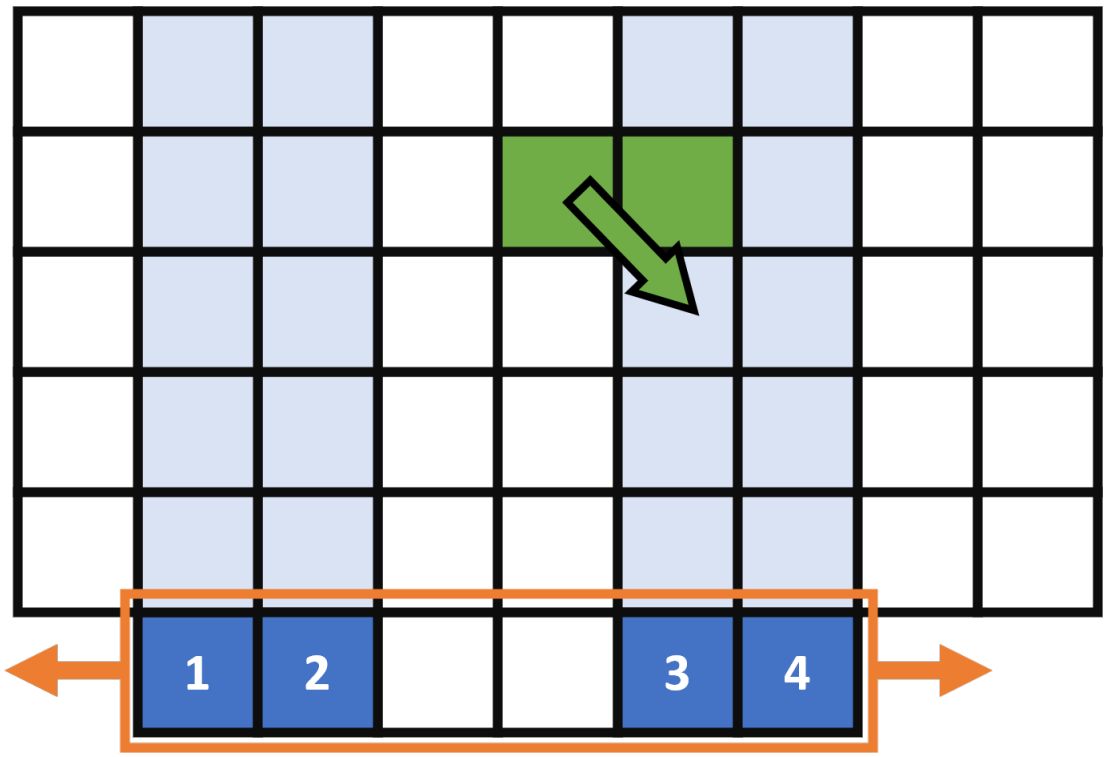

Figure 2.1: A subsection of a single case of the Active Categorical Perception Task. The agent, visualized by the orange box, can move either to the left or right as indicated by the orange arrows. The four sensors, labelled 1,2,3, and 4, can each see directly upwards as indicated by the light blue shading. A small block with width 2 is falling towards the agent and moving to the left, as indicated by the green arrow. At the current point in time, sensor 3 is the only sensor that would return a signal, while on the next update, assuming that the agent did not move, both sensors 3 and 4 would return a signal.

To study generalization, I use a variety of gap widths and small and large block sizes. The combinations of block sizes used during evolution and for testing at the end of evolution are

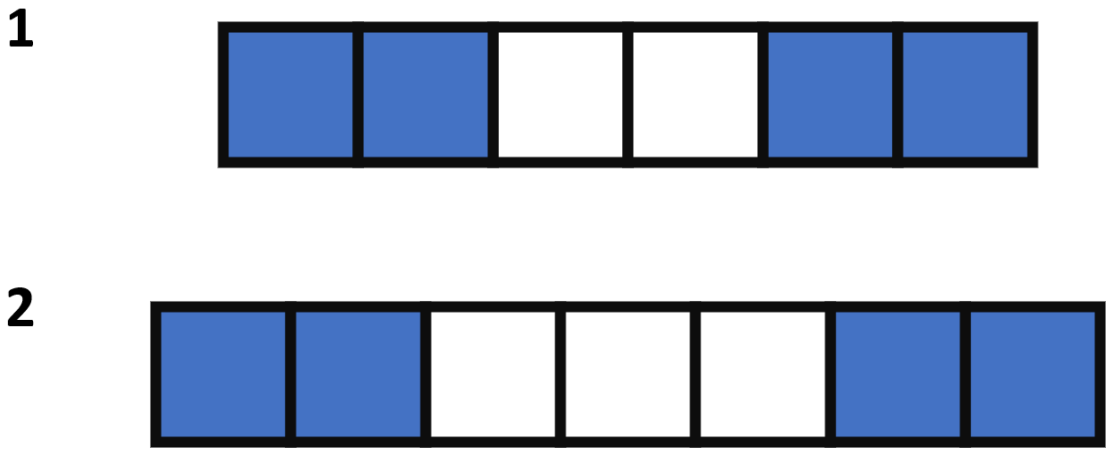

Figure 2.2: The two agent types used for generalization trials. The agent labelled 1 has the standard gap width of 2 units, while the agent labelled 2 has the modified gap width of 3 units. Blue squares indicate sensors, while the white squares indicate the gap in the middle.

listed in Table 2.1. For all other experiments, I use the standard gap width of 2 and blocks of size 2 and size 4.

Regardless of the unique combination of gap width and block size, the agent's objective in the ACP task is unchanged: the agent either makes the correct or incorrect decision to avoid or catch each block. Following from previous work, I comprehensively tested each agent over a range of decisions based on the starting conditions of the block. Given the width of the world, the block could start at 16 unique locations relative to the agent which always started with its leftmost sensor at the position with index 0. Given that the blocks are either small or large, move to the left or to the right, and can start in 16 unique starting positions, there are 64 unique combinations which are comprehensively evaluated both during evolution and testing. Agents need to observe the block for multiple updates to make the correct decision; this leads to the integration of information over time which is aided by the creation of internal models and representations.

Table 2.1: Conditions for Evolution and Testing of block sizes to catch (small) and avoid (large) as well as the size of the gap between the sensors.

condition	gap size	during evolution		during testing	
		small block size	large block size	small block size	large block size
2v4/2/1v5	2	2	4	1	5
2v4/3/1v5	3	2	4	1	5
1v5/2/2v4	2	1	5	2	4
1v5/3/2v4	3	1	5	2	4
3v5/2/2v6	2	3	5	2	6
3v5/3/2v6	3	3	5	2	6
2v6/2/3v5	2	2	6	3	5
2v6/3/3v5	3	2	6	3	5

2.3.3 Representations and the neuro-correlate $\mathcal{R}$

Representations are defined here as the information that an agent has about its environment (in this case the ACP task) that is not currently reflected in the sensors. This definition limits representations to consist of information that has been integrated into the agent's memory, and thus representations are a reflection of preexisting mental models that the agent has constructed. The information-theoretic measure $\mathcal{R}$ (where $\mathcal{R}$ stands for Representation) is designed to quantify the amount of representations present under this definition, shown in equation 2.1. The measurement of $\mathcal{R}$ is the result of an entropic calculation over environmental or world states (W), memory or brain states (B), and sensor states (S).

$$\mathcal{R} = H(W:B|S) = H(W:B) - I(W:B:S)\ . \tag{2.1}$$

This relationship can also be more easily visualized by the information-theoretic Venn diagram, seen in Figure 2.3).

$\mathcal{R}$ is calculated in practice using a sampling technique. The states of all three variables W, B, and S are recorded during the lifetime of the agent, and the approximate value of $\mathcal{R}$

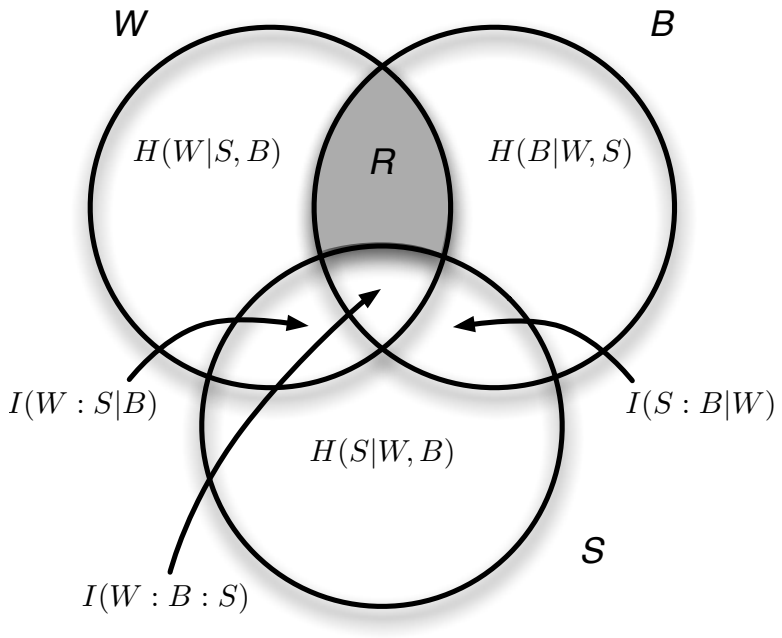

Figure 2.3: Venn diagram of entropies and informations for the three random variables W, S, and B, describing the world, sensor, and agent internal (brain) states. The representation $\mathcal{R} = H(W:B|S)$ is shaded.

is found by applying Shannon entropy calculations to the probability of these recorded states. Although the brain and sensor states are easily determined by examination of the Markov Brain's sensor and hidden states, the environmental states require some consideration. In order for $\mathcal{R}$ to be an effective measure and be useful for $\mathcal{R}$-augmentation, the world states need to be carefully chosen by the practitioner. The variables chosen as world states attempt to capture the concepts whose knowledge are most likely predictive of the agent's performance, or are the most useful for the agent to understand in order to solve the task. In a slight variation from the original formulation of [Marstaller et al., 2013], I only use three of the environmental concepts proposed there, as these should be sufficient for completing the task: the size of the block (large or small), the direction of block movement (left or right), and the position of the block relative to the agent (left or right). Preliminary work suggested that that agents typically do not evolve representations about whether they are currently under the block (the foruth variable used in [Marstaller et al., 2013]), and so I omit it in this work.

2.3.4 Evolutionary Algorithm

I test the performance of each agent in each generation using the active categorical perception task. Specifically, since the world is 16 units wide and there are two different sized blocks that could fall into two directions, each agent can experience at most 64 different start conditions. Agents are tested using all these possible start conditions and thus can make in total 64 correct catch-or-avoid decisions. I use the number of correct decisions C and incorrect decisions I to define the fitness function as:

$$W = 1.10^{(C-I)} \tag{2.2}$$

I use the same formula for augmenting the genetic algorithm with the measurement of $\mathcal{R}$ as defined in [Schossau et al., 2015]:

$$W = 1.10^{(C-I)}(1 + \frac{R}{R_{max}}) \tag{2.3}$$

where R_{max} is defined by the maximal size of $I(W : B)$ (here 3.0).

I use roulette wheel or fitness-proportional selection [Thomas, 1996] in conjunction with the calculated fitness values to determine which organisms shall be parents for the next generation of agents. I apply the mutations as described in Section 2.3.1 to the selected parents, and use these mutated offspring to construct a new generation of organisms with the same number of organisms as the previous generation.

2.3.5 Robustness to Noise

To evaluate an evolved agent's robustness towards noise, I test all agents on all possible start conditions of the active categorical perception task. Unlike the execution of the task as used during evolution, at every temporal update of the environment I add different levels of noise

to the agent's sensor values. Specifically, each of the four sensors had an independent chance to not show the actual environment but contain a random bit.

2.3.6 Generalization

Generalization in machine learning, for example in classification tasks, is often described as the ability to not only perform well on the training data, but to also perform well on test data that was not used for training [Russell and Norvig, 2016], but the concept of "generalization" is really more general than that. Here, I instead think of the ability to generalize as the capacity of a mental model to be applied to more than a specific case so that the agent can extrapolate to other similar tasks [Johnson-Laird, 1980]. To be more specific, a classifier trained to distinguish hand written numerals for example might only be able to answer correctly when confronted with images from the training set. Ideally, it should also have the ability to generalize to classify numerals in images it has not seen before. Even better would be the ability to perform related but new tasks (this is similar to transfer learning [Pan and Yang, 2009]). An example of this ability to complete new tasks would be asking the classifier trained on predicting numerals to now distinguish letters. While for typical classifiers this ability to tackle new tasks is an extremely difficult problem, generalizing is something mammalian brains seem to be able to do easily.

For the active categorical perception task, this generalization to a new task (also called out-domain generalization) can be examined by testing agents on blocks of sizes that have not been seen during evolution. For example, agents with a gap size of 2 between the sensors were evolved to catch size 2 and avoid size 4 are then tested to see how well they can catch size 1 and avoid size 5. I evolved the agents on the following small/large block pairings: 2/4, 1/5, 3/5, and 2/6. The first two and the last two sets of sizes were reversed for testing (i.e., agents evolved on 2/4 were tested on 1/5, and agents evolved on 2/6 were tested on 3/5). I tested all block size combinations with both the gap width sizes, 2 and 3. For a complete enumeration of the tested conditions see Table 2.1.

2.4 Results

I first test that using a GA augmented by rewarding $\mathcal{R}$ increases adaptation as shown in [Schossau et al., 2015]. I use this repetition to also test that our modified definition of the environment and the different implementation of measuring $\mathcal{R}$ does not adversely affect the positive effect $\mathcal{R}$ has on adaptation. I find that for agents evolved to catch blocks of size 2 and avoid blocks of size 4 with a gap width of 2, as expected, using $\mathcal{R}$ (see Equation 2.3) allows the genetic algorithm to evolve faster and to produce better solutions on average (see Figure 2.4) as opposed to not using $\mathcal{R}$ (equation 2.2).

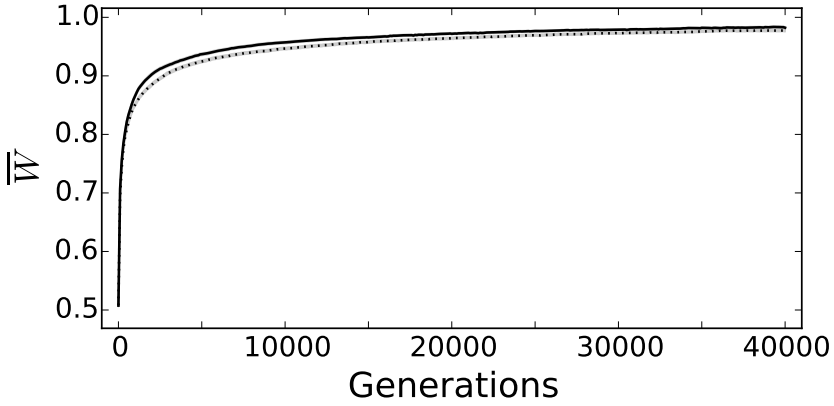

Figure 2.4: Average performance ($\bar{W}$ over generations for agents evolved without using $\mathcal{R}$ (dotted line) and with augmenting the GA (solid line). Averages are computed for agents on the line of descent over 800 replicates. The standard error of the means are shown in grey, although due to the high replication count the standard error is negligible. The task performance of agents evolved using a GA augmented with $\mathcal{R}$ and a standard GA are significantly different when tested using Wilcoxon-Ranked-Sum test ($p = 1.88e - 67$). Here I evolved agents to catch blocks of size 2 and avoid blocks of size 4, with a gap width between the sensors of size 2.

With the baseline increase in performance established in Markov Brains with Deterministic Logic Gates, I then confirmed that the average performance at the end of evolution was higher in Markov Brains evolved with $\mathcal{R}$-augmentation than those evolved without, for a variety of gate types. This result is shown for Probabilistic Logic Gates (Figure 2.5), GP Gates (Figure 2.6), ANN Gates (Figure 2.7), and NEAT Gates (Figure 2.8). For all gate types but the Probabilistic Logic Gates, the average performance is greater for Markov Brains evolved

under $\mathcal{R}$-augmentation than those without for at least 80% of the 10,000 generations.

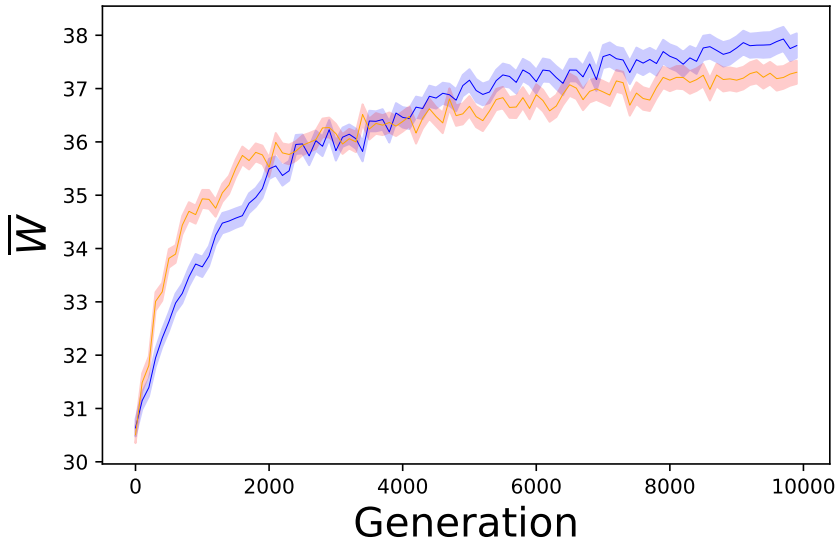

Figure 2.5: Task performance in Markov Brains with probabilistic logic gates produced by augmented and unaugmented GAs. The blue line depicts results from the lines of descent of GAs using $\mathcal{R}$-augmentation, while the orange line comes from unagumented GAs. Each line is the performance averaged over 500 replicate experiments. The shaded area around each line is the standard error.

I also confirmed that for all other agent configurations and objectives such as different sized blocks and different gap sizes (see Table 2.1), I observe the same phenomenon: using $\mathcal{R}$ to augment the GA improved performance across all tested conditions (see Figure 2.9). The difference in performance of agents evolved with a GA augmented with $\mathcal{R}$ and those evolved without augmentation is statistically significant (Mann-Whitney U, $p < 0.01$ for all conditions), and the mean performance of those evolved in a GA augmented with $\mathcal{R}$ was always higher than those evolved in the control conditions.

Different environmental conditions may affect the performance of the GA, as well as the magnitude of impact that augmentation with $\mathcal{R}$ had on the GA. I test if augmenting the GA with $\mathcal{R}$ in any way compromises the ability of the evolved agents to generalize. To demonstrate the agent's ability to generalize, agents evolved in one environment are tested on a new task where blocks had different sizes, but with a constant gap size so that the agent's internal understanding of the gap width between sensors was not altered. I find no significant

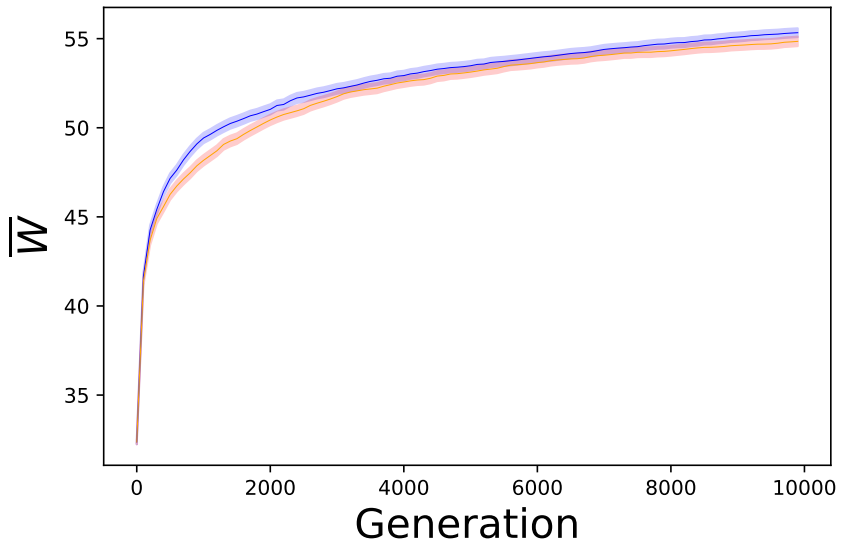

Figure 2.6: Task performance in Markov Brains with GP gates produced by augmented and unaugmented GAs. The blue line depicts results from the lines of descent of GAs using $\mathcal{R}$-augmentation, while the orange line comes from unagumented GAs. Each line is the performance averaged over 500 replicate experiments. The shaded area around each line is the standard error.

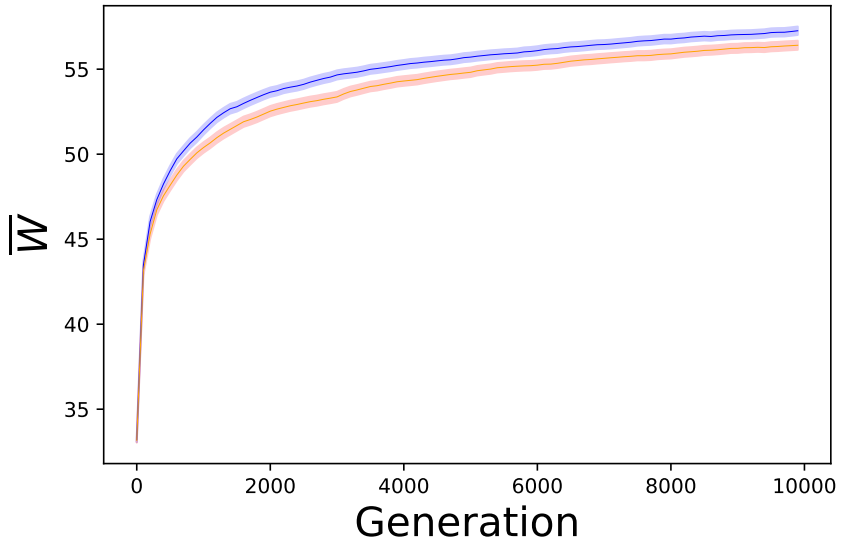

Figure 2.7: Task performance in Markov Brains with ANN gates produced by augmented and unaugmented GAs. The blue line depicts results from the lines of descent of GAs using $\mathcal{R}$-augmentation, while the orange line comes from unagumented GAs. Each line is the performance averaged over 500 replicate experiments. The shaded area around each line is the standard error.

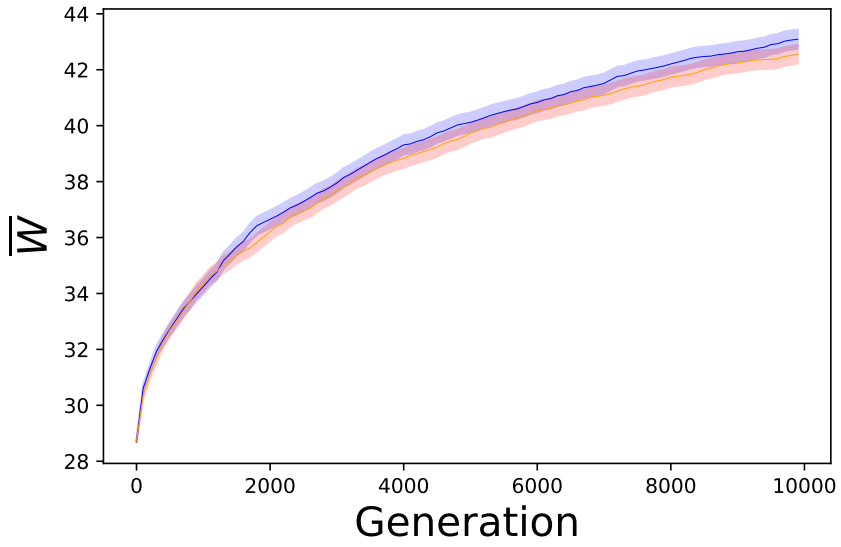

Figure 2.8: Task performance in Markov Brains with NEAT gates produced by augmented and unaugmented GAs. The blue line depicts results from the lines of descent of GAs using $\mathcal{R}$-augmentation, while the orange line comes from unagumented GAs. Each line is the performance averaged over 500 replicate experiments. The shaded area around each line is the standard error.

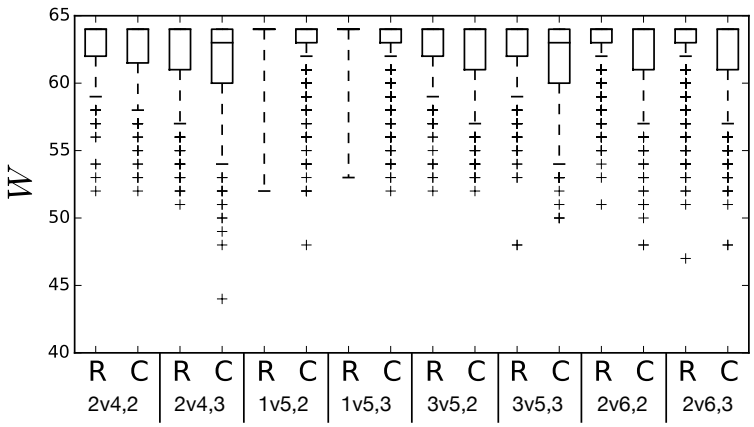

Figure 2.9: Performance of agents evolved under different evolutionary conditions after 40,000 generations. Each column shows the distribution of fitness as a box plot indicating the mean, the 25th quartile, the 75th quartiles, and outliers as '+'. R indicates an evolutionary experiment where the GA was augmented by using $\mathcal{R}$; C labels the control. The box plots show the results for 8 different experimental conditions using combinations of different block sizes (2v4, 1v5, 3v5, 4v6) and gap widths (2,3).

loss in the ability to generalize across different experimental conditions (see Figure 2.10).

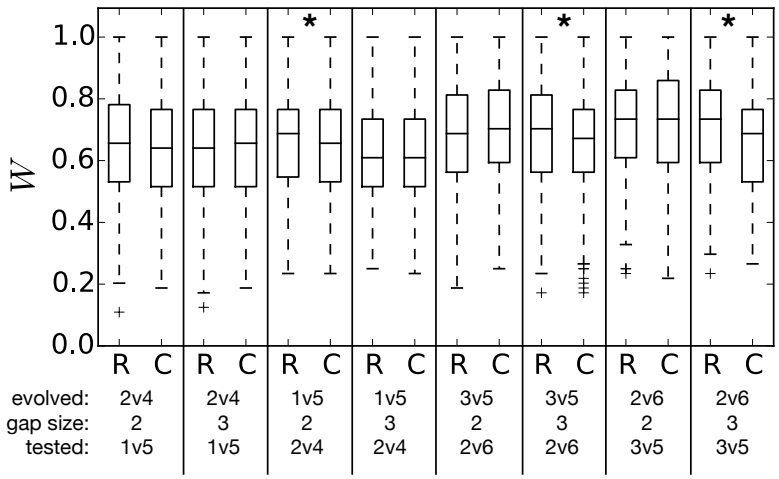

Figure 2.10: Performance of evolved agents after 40,000 generations tested on different conditions than they were evolved in. Each column shows the distribution of performance as a box plot indicating the mean, the 25th quartile, the 75th quartiles, and outliers as '+'. R indicates an experiment where the GA was augmented by using $\mathcal{R}$, C labels the control. As defined in Table 2.1 I used 8 different environments to evolve and consequently test them. Significant differences (Mann-Whitney U, $p < 0.01$) are marked with *. In each of the significant cases the mean tested performance was greater when evolved with $\mathcal{R}$-augmentation than in the control (unaugmented) condition.

Whenever I observe a significant difference in the agents ability to generalize, agents evolved in a GA that are augmented with $\mathcal{R}$ have a mean performance greater than those that were not evolved with $\mathcal{R}$.

Now that I have shown that using $\mathcal{R}$ is not detrimental but might even be slightly beneficial to the agents ability to generalize, I ask how $\mathcal{R}$ affects robustness. To that end, I test agents evolved in a GA with and without augmentation using $\mathcal{R}$ under different levels of noise. Specifically, each sensor had a chance to show a random 0 or 1 instead of the original signal, and I modify the likelihood for the bit to flip over the range from 0.0 to 0.2 in intervals of 0.01. I find that the $\mathcal{R}$-augmented and unaugmented GAs produce networks with no consistent difference in performance for each of the tested conditions (see Figure 2.11).

This result, where robustness is the same regardless of augmented or unaugmented GA, shows that augmentation by using $\mathcal{R}$ does not impede the GA's ability to select for agents

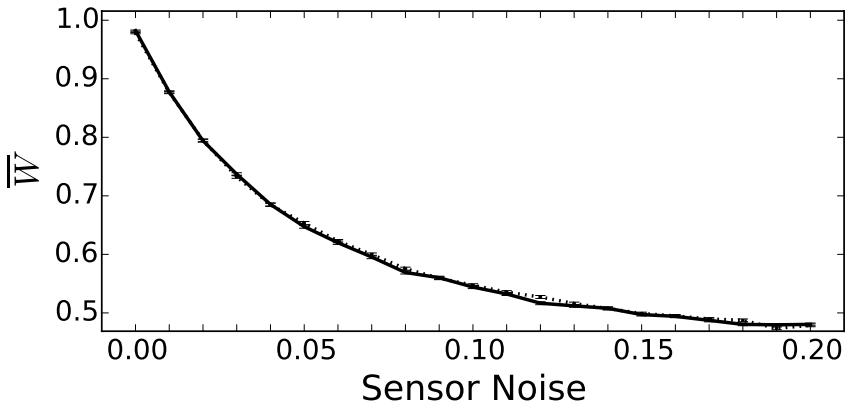

Figure 2.11: Mean performance of agents evolved with (solid line) and without (dashed line) augmenting the GA with $\mathcal{R}$. The performance ($\bar{W}$) is from agents from 800 replicate experiments after 40,000 generations of evolution tested over different levels of sensor noise. Error bars indicate the standard error. The results were not significantly different when tested with a Wilcoxon Rank-Sum test. These agents were evolved to catch blocks of size 2 and avoid blocks of size 4 and had a gap width of 2.

that are robust to sensor noise under these conditions. This experiment presents agents with sensor noise for the first time in the test environment, however the agents never experienced noise during evolution. I now repeat the evolutionary experiment but also apply the different levels of noise during evolution. I find that agents evolved under noise while their GA was augmented with $\mathcal{R}$ perform significantly better than those evolved without the augmentation by $\mathcal{R}$ (see Figure 2.12) in non-noisy environments.

These results from evolution under noise show an interesting feature of augmenting a GA with $\mathcal{R}$. In the presence of noise, agents typically face a more difficult environment to master than without noise. At the same time, noise reduces the accuracy of fitness as a measurement of capability because noise affects the agent's ability to predict the true state of the environment. The noise should, at least in theory, also affect the calculation of $\mathcal{R}$ since it takes the sensor states into account as well. Surprisingly, the presence of noise during evolution does not hinder the improvements to task performance provided by using $\mathcal{R}$-augmentation.

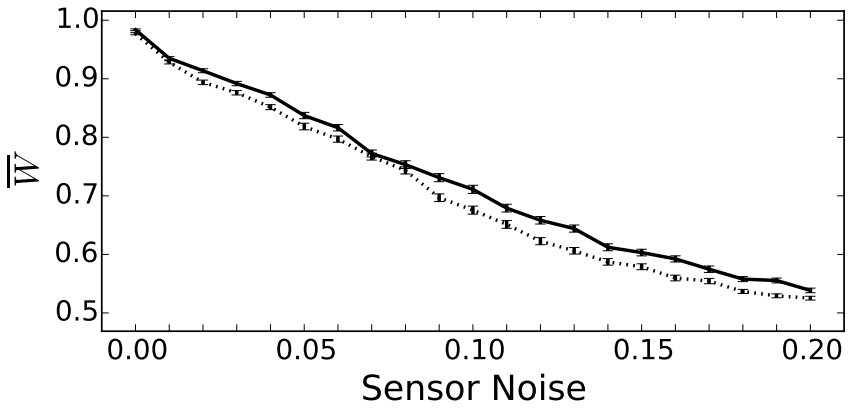

Figure 2.12: Mean performance of agents evolved under different levels of noise (x axis) with (solid line) and without (dashed line) augmenting the GA with $\mathcal{R}$. The performance ($\bar{W}$) is from agents from 350 replicate experiments for each level of noise, after 40,000 generations of evolution. Error bars indicate the standard error. The results were significantly different when tested with a Wilcoxon Rank-Sum test ($p = 5.95e - 5$). These agents were evolved to catch blocks of size 2 and avoid blocks of size 4 and had a gap width of 2.

2.5 Discussion

Augmenting a genetic algorithm by using the information-theoretic measure $\mathcal{R}$ accelerates evolutionary adaptation. It was previously not known whether this result would be adaptable to other computational components. In this chapter I show that $\mathcal{R}$-augmentation also works with a variety of different computational gates in the Markov brain system. While it was previously unclear if this method incurs trade-offs, such as causing the evolved solutions to be less robust or to have a diminished ability to generalize to a new task, I show that comparing the robustness of solutions evolved with and without the augmentation of $\mathcal{R}$ reveals no significant trade-off. However, when also evolving under the influence of sensor noise, I find that augmenting the GA with $\mathcal{R}$ results in more capable solutions than those evolved by an unaugmented GA.

While the direct mechanism for this improvement in performance regardless of sensor noise is not clear, I hypothesize that any negative effect of noise on the fitness measurement is compensated by the $\mathcal{R}$ term in the augmented fitness function (see Equation 2.3). In the case where $\mathcal{R}$-augmentation is used, the GA is able to take a second measure ($\mathcal{R}$) into account

and is no longer dependent only on the noisy task performance measurement.

As an alternative explanation of the increased performance under varying levels of sensor noise, I hypothesize that using $\mathcal{R}$ to augment the fitness of the GA results in the selection of agents that experience more unique conditions of the task (i.e., explore the environment) over those that do not explore the environment to the same extent but have the same fitness. Specifically, for an agent to refine its internal representation (have a higher value of $\mathcal{R}$), the agent needs to explore more of the environment. For an agent to improve $\mathcal{R}$, maybe that agent needs to take into account additional observations that they might have otherwise discarded. Of course, it is precisely these additional observations that might mitigate the effect of sensor noise. Consequently, $\mathcal{R}$ does not directly affect fitness but encourages the GA to select agents that are more robust to noise indirectly, which ultimately produces agents that perform better.

I also found that the agents' ability to generalize to a new task is mostly unaffected, or even improved, by augmenting the GA with $\mathcal{R}$. In 3 out of the 8 generalization experiments, $\mathcal{R}$-augmented GAs produced networks with significantly higher performance on the new version of the task, although the effect size is not large. In the other 5 the performance of the networks produced by $\mathcal{R}$-augmented GAs is not significantly different than the performance of networks produced by unaugmented GAs. In other words, the networks produced by $\mathcal{R}$-augmented GAs generalize at least as well as or better than networks produced by unaugmented GAs.

In all experiments I evolved Markov Brains to solve an active categorical perception task, and used varying types of computational components. Representations are arguably an important aspect of general purpose intelligence, but these experiments do not allow us to extrapolate the effect of $\mathcal{R}$-augmentation to other cognitive abilities or other structurally different cognitive systems such as RNNs. However, given that $\mathcal{R}$-augmentation is effective when applied to architectures with a variety of computational components, I can be confident that this method can be effective when applied to artificial cognitive systems that vary in

structure.

The computation of $\mathcal{R}$ depends on the specification of variables identifying world state that a well-performing agent should know about. If the experimenter chooses these variables poorly, the resulting $\mathcal{R}$ could be irrelevant to the task. As an example the blocks in our experiment had two distinct colors, but the objective remains to catch small and avoid large blocks regardless of color, selecting for knowledge about the color will probably not improve adaptation. In principle the relative value or cost of such counterfactual representations can and should be tested.

2.6 Conclusions

The work described here reaffirms the benefits of $\mathcal{R}$ as a neuro-correlate capable of accelerating rates of adaptation, and prompts several new avenues of research to follow. I propose that future work adapting $\mathcal{R}$-augmentation to machine learning is a worthwhile endeavor. Machine or deep learning systems seem to struggle with creating internal models and representations, and $\mathcal{R}$-augmentation could aid in the development of representations for those systems. Furthermore, the use of $\mathcal{R}$ does not compromise the ability of an evolved agent to generalize or to be robust against sensor noise, which are both important properties for AI systems. Unfortunately, performance optimization by a GA is very different from gradient descent or other optimization methods in deep learning. As such, $\mathcal{R}$ cannot simply be applied in the context of machine learning, even though these systems appear similar at first glance. Further research will be necessary to adapt $\mathcal{R}$-augmentation to machine learning or deep learning.

Perhaps the most interesting lesson learned is that $\mathcal{R}$-augmentation affects how brains evolve to deal with noisy sensor data. While using $\mathcal{R}$ made no difference to sensor noise robustness when agents were evolved without noise, evolution under noise resulted in significantly better adaptation when the GA was augmented with $\mathcal{R}$. This implies two things: evolving representations under noise might be a harder challenge than performing under

noisy situations after normal (i.e., noise-free) evolution, and the augmentation therefore is advantageous as it increases the ability of the agent to adapt to the noisy situation.

Augmenting the GA with $\mathcal{R}$ evolves better performing agents under noise than the unaugmented GA, which implies that the $\mathcal{R}$-augmentation process makes up for a lack of information presented to the agent. Additionally, because the noise in the sensors will affect the calculation of $\mathcal{R}$, the result of $\mathcal{R}$-augmentation performing better under noise implies that the measurement of $\mathcal{R}$ is robust to incomplete information. This implication opens up an interesting line of thinking: How can I use this robustness to incomplete information to increase the performance of the $\mathcal{R}$-augmentation process? I propose that instead of having noise cause the incomplete information, using a reduction of the cases presented to the agent would have a similar effect. This evaluation on only a subset of cases would result in a less optimal assessment of fitness, which could be counteracted by using $\mathcal{R}$ as a co-evolutionary factor at the same time. Obviously using $\mathcal{R}$ as the means for augmentation imposes an extra computational cost, but that would now be offset by using fewer test cases. Using $\mathcal{R}$ to offset a reduction in test cases is an interesting extension of this work that I seek to investigate further.

Chapter 3

The Evolution of Representations in Genetic Programming Trees

This chapter was originally published as "Kirkpatrick, D., & Hintze, A. (2020). The Evolution of Representations in Genetic Programming Trees. In Genetic Programming Theory and Practice XVII (pp. 121-143). Springer, Cham." Used with permission. Minor changes have been made to fit the tone and style of this dissertation.

3.1 Abstract

Artificially intelligent machines have to explore their environment, store information about it, and use this information to improve future decision making. As such, the quest is to either provide these systems with internal models about their environment or to imbue machines with the ability to create their own models - ideally the later. These models are mental representations of the environment, and research has previously shown that neuro-evolution is a powerful method to create artificially intelligent machines (also referred to as agents) that can form said representations. Furthermore, previous work has shown that one can quantify representations and use that quantity to augment the performance of a genetic

algorithm. Instead of just optimizing for performance, one can also positively select for agents that have better representations. The neuro-evolutionary approach, that improves performance and lets these agents develop representations, works well for Markov Brains, which are a form of Cartesian Genetic Programming network. Conventional artificial neural networks and their recurrent counterparts, RNNs and LSTMs, are however primarily trained by backpropagation and not evolved, and they behave differently with respect to their ability to form representations. When evolved, RNNs and LSTMs do not form sparse and distinct representations, they "smear" the information about individual concepts of the environment over all nodes in the system. This ultimately makes these systems more brittle and less capable. The question I seek to address, now, is how can we create systems that evolve to have meaningful representations while preventing them from smearing these representations? I look at genetic programming trees as an interesting computational paradigm, as they can take a lot of information in through their various leaves, but at the same time condense that computation into a single node in the end. I hypothesize that this computational condensation could also prevent the smearing of information. Here, I explore how these tree structures evolve and form representations, and I test to what degree these systems either "smear" or condense information.

3.2 Introduction

One of the most promising approaches in the quest for general purpose artificial intelligence (AI) is neuro-evolution [Yao, 1999, Floreano et al., 2008, Stanley et al., 2019]. The idea is to use a genetic algorithm to optimize the AI that controls an embodied agent to solve one or many cognitive tasks. Many different AI systems have been proposed for this task, such as Artificial Neural Networks (ANNs) and their recurrent versions (RNNs) [Russell et al., 2003], Genetic Programs (GPs) [Koza, 1994], Cartesian Genetic Programming (CGP) [Miller, 2011], Neuro Evolution of Augmenting Topologies (NEAT) [Stanley and Miikkulainen, 2002], and

Markov Brains (MBs) [Hintze et al., 2017], among many others. One of the most pressing problems is that AI have difficulty generating internal models about the environment they act in. Humans seem to be particularly good at creating these models for themselves, but we struggle greatly when it comes to imbuing artificial systems with such models, also called representations[1]. It even has been argued that one should not even try to do so and instead create "intelligence without representation" [Brooks, 1991]. However, previous work found that neuro-evolution is very capable of solving this problem of creating structures to store representations.

To prove that a system indeed has representations, previous work introduced an information theoretic measure $\mathcal{R}$ that quantifies exactly how much information the internal states of a system store about its environment independent of the information coming directly from its sensors [Marstaller et al., 2013]. Over the course of evolution Markov Brains stored more information about the environment necessary to solve complex tasks. They did so, by associating specific hidden nodes with different environmental concepts. These sparse and distributed representations are believed to be similar to how humans for example store information: Not all neurons fire, but certain areas perform computations based on specific firing patterns of other neurons. I also found that RNNs, when evolved to perform a task that requires memory, form less sparse representations and smear the information across all their hidden states in contrast to MBs.

However, it remains to be shown that genetic programming (GP) or systems similar to GP (such as CGPs) also evolve to solve tasks that require representations, to what extent they evolve the ability to form representations, and to what degree they smear their representations.

While it is academically interesting to find how much a system knows about its envi-

[1]Observe that the term representations in computer science sometimes also refers to the structure of data, or how an algorithm, for example, is encoded. I mean neither, but instead use the term representation to be about the information a cognitive system has about its environment as defined in Marstaller et al. 2013 [Marstaller et al., 2013]. The term representation, as I use it, is adapted from the fields of psychology and philosophy

ronment, quantifying representations has an application for genetic algorithms. Similar to multiple objective optimization [Zhou et al., 2011] which improves at least two different qualities of a system at the same time, one can not only use the performance of an agent but also use the degree to which the system has developed representations to define the mean number of offspring in a genetic algorithm (GA) for augmentation of the GA [Schossau et al., 2015]. While it has been shown how this helps to optimize Markov Brains and RNNs during evolution, again it is not known if the same is true for genetic programming systems. Lastly, representations can not only be measured for the whole system but also for individual sub-components. By testing all sub-components versus all concepts in the environment, I can pinpoint where these representations are exactly stored in the system. This results in a matrix defining the amount or information each element has about each concept. I found earlier [Hintze et al., 2018] that substrates differ in how distributed or condensed these representations are. Markov Brains create sparsely distributed representations, while RNNs smear the knowledge about the environment across all nodes. In addition, I found that for Markov Brains, RNNs, and LSTMs [Hochreiter and Schmidhuber, 1997] that smeardness and robustness correlate negatively. Perhaps the difference in representation structure between Markov Brains and RNNs is not surprising when one considers the topological features of these systems. RNNs receive their inputs and sequentially propagate this information through one layer after the other by connecting each node from one layer to all nodes to the next. While it obviously ensures that the information can reach all components it is presumably also preventing representations to become local and sparse. Markov Brains, on the other hand, start with sparse connections and only retain additional ones if there is an evolutionary benefit at the time of addition. This results in sparsely interconnected graphs, which might be the reason why these kinds of networks evolve to have sparse and condensed representations.

Genetic programming trees present a great tool to evolve and fit almost arbitrary mathematical functions to data. They can take a lot of different inputs, perform computations on them, and due to their tree structure, condense the result of all those computations into

a single result at the root of the tree. It stands to reason that this topology provides an advantage to the question of condensing representations as well. These trees can be as wide as the input layers of an RNN, but do not have the property of further distributing the information at every step (see Figure 3.1 for a comparison of topologies and representational smearedness across different kinds of computational substrates).

The genetic programming trees, however, have the disadvantage that they do not have an intuitive way to implement recurrence. One way of dealing with the exact problem of forming memory has been addressed by adding *indexed memory* and additional computational nodes into the tree that can read from and write to said memory [Teller, 1994]. Similarly, the nodes that perform the read and write operations, or push and pop in the case of a memory stack, can themselves be evolved by genetic programming trees. These trees implement commands that can then be included into linear forms of genetic programming [Langdon, 1995]. Finally, the genetically evolvable push programming language [Spector and Robinson, 2002] provides a more complex solution for memory stored in stacks, specifically for linear genetic programs. While these methods all solve the problem of adding memory to the system, they essentially deviate from the idea of using the property of the tree to potentially condense information. Therefore, I am testing two alternative ways to create recurrence in genetic programming trees. I test if, as a consequence, the tree topology provides any advantage with respect to the ability to evolve sparsely condensed or smeared representations.

3.3 Material and Methods

3.3.1 Representations and the neuro-correlate $\mathcal{R}$

The neuro-correlate $\mathcal{R}$ seeks to quantify the amount of information an agent has about its environment that is not currently provided by its sensors. The information-theoretic Venn diagram (see Figure 3.2) illustrates the relation between environment or world states (W),

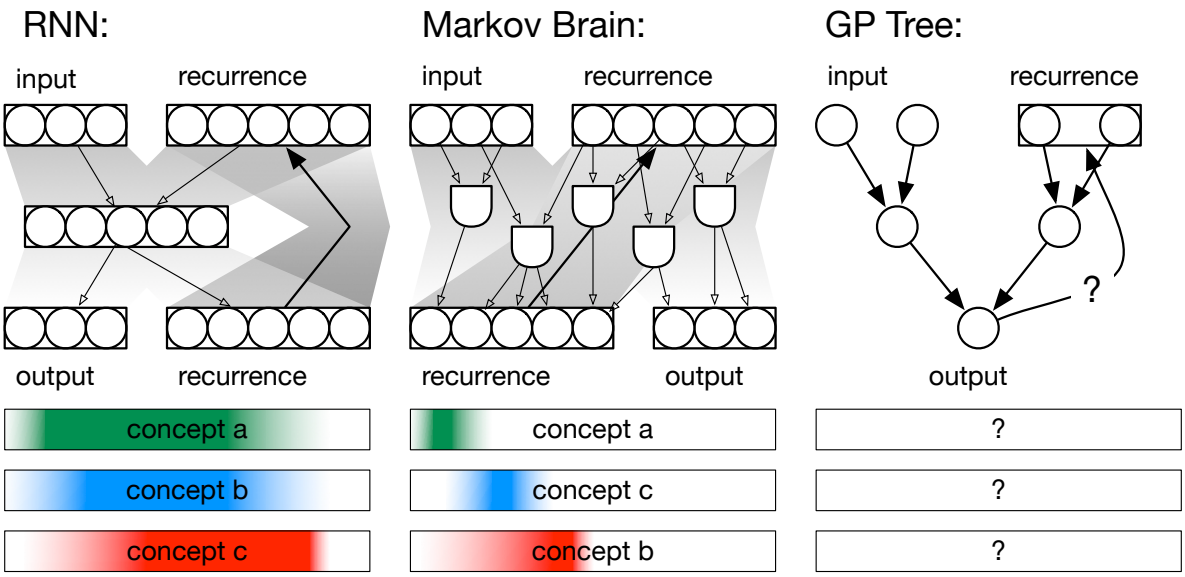

Figure 3.1: Three different cognitive architectures - RNN, Markov Brain, and GP tree - and how they distribute or condense representations. RNNs have a joint input and recurrence layer, feed into arbitrary many hidden layers of arbitrary size, compute outputs which are also recurred back to the input layer. These networks tend to smear representations about different context overall recurrent nodes. The red, blue, and green color bars illustrate that. Markov Brains use inputs and hidden states as inputs to the logic gates or to other kinds of computational units. The result of the computations creates recurrent states and outputs. These networks tend to form sparse condensed representations, illustrated by narrower red, green, and blue bands. GP trees use inputs and potentially recurring information and combine them over several computational nodes into a result. For these structures, it is not clear to which degree they are smearing or condensing representations. It is also not obvious how to recur them, illustrated by the "?".

memory or brain states (B), and sensor states (S) and defines $\mathcal{R}$ to be quantified as:

$$\mathcal{R} = H(W : B|S) = H(W : B) - I(W : B : S) \ . \tag{3.1}$$

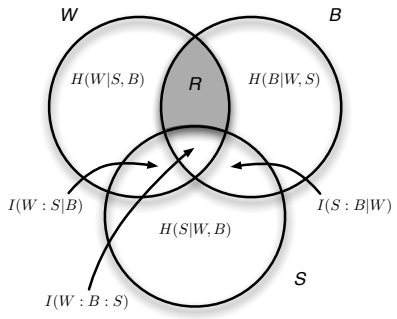

Figure 3.2: Venn diagram of entropies and information for the three random variables W, S, and B, describing the world, sensor, and agent internal (brain) states. The representation $\mathcal{R} = H(W:B|S)$ is shaded.

In order to actually measure $\mathcal{R}$ one needs to record the states of all three random variables. In the case of Markov Brains, this means the brain and sensor states are just the state of the hidden and input nodes. The world states, however, are not that obvious. In fact, they need to be defined and well-chosen by the experimenter and depend highly on the environment. Here I use a block catching task (active categorical perception, see 3.3.3) and number comparison task (number discrimination, see 3.3.4), and thus use the relevant concepts of these environments to define the world states. Here, for the block catching task, I chose the size of the block, the direction of the block falling, and whether the block is to the left or right of the agent. This is different from Marstaller [Marstaller et al., 2013] where they also measured if the block is currently above or next to the agent. It was observed that agents typically do not evolve representations about hitting or missing the block, and thus this state was omitted. For the number comparison task, I chose the actual numbers to be the world states. As such, I ask to what degree the agent evolves representations about each number explicitly. In addition, I also defined a state to represent whether the first or second number was larger, to see if the agent is able to represent the difference between the two numbers.

With regards to the actual computation of $\mathcal{R}$, a reduction was found that decreases the number of steps needed to compute $\mathcal{R}$. Where the original equation 3.1 expanded in practice to

$$\mathcal{R} = H(W) + H(B) + H(S) - H(W, B, S) - I(S : W) - I(S : B) \ , \tag{3.2}$$

I found a reduction to only four entropy calculations:

$$\mathcal{R} = H(S, W) + H(S, B) - H(S) - H(W, B, S) \ . \tag{3.3}$$

The proof for this reduction is as follows:

Proof. I start with the original formula for R from [Marstaller et al., 2013]:

$$\mathcal{R} = H_{Corr} - I(S : W) - I(S : B),$$

where H_{Corr} is defined as

$$H_{Corr} = H(W) + H(B) + H(S) - H(W, B, S).$$

This expands to

$$\mathcal{R} = H(W) + H(B) + H(S) - H(W, B, S) - I(S : W) - I(S : B).$$

I then use the formula for conditional information from [Marstaller et al., 2013]

$$I(X : Y) = H(X) + H(Y) - H(X, Y)$$

to expand

$$I(S:W) = H(S) + H(W) - H(S,W)$$

and

$$I(S:B) = H(S) + H(B) - H(S,B).$$

Substituting these leads to

$$\begin{aligned}\mathcal{R} =& H(W) + H(B) + H(S) - H(W,B,S) - (H(S) + H(W) - H(S,W)) \\ & - (H(S) + H(B) - H(S,B)).\end{aligned}$$

By distributing the negative signs I get

$$\begin{aligned}\mathcal{R} =& H(W) + H(B) + H(S) - H(W,B,S) - H(S) \\ & - H(W) + H(S,W) - H(S) - H(B) + H(S,B).\end{aligned}$$

The positive and negative entropies cancel, leaving

$$\mathcal{R} = -H(S) - H(W,B,S) + H(S,W) + H(S,B).$$

A slight reordering gives the final equation,

$$\mathcal{R} = H(S,B) + H(S,W) - H(S) - H(W,B,S).$$

□

As the reduced equation (3.3) requires fewer mathematical operations than the original equation (3.2), there is less chance for computational error and round-off in the reduced equation, which makes it more accurate in practice than the original. In addition, the reduced equation is more computationally efficient to calculate.

3.3.2 Smearedness of Representations

While the neuro-correlate $\mathcal{R}$ calculates the amount of mental representations as a whole, it does little to examine the structure or layout of said representations. Recent work [Hintze et al., 2018] proposes a new measure, referred to as smearedness, to work around this deficiency. Smearedness looks at the amount of representation that each memory state in the brain has about each individual concept in the environment, and takes a pairwise minimum to see how much the representations are spread across the different nodes. The equation for smearedness is seen in equation 3.4, where M is the measure of the representation in a specific node and concept, taken over all nodes i and for all combinations of concepts j and k.

$$\mathcal{S}_N = \sum_i \sum_{j>k} min(M_{ji}, M_{ki}) \tag{3.4}$$

3.3.3 Active Categorical Perception Task

In order to augment the performance of a genetic algorithm using the neuro-correlate $\mathcal{R}$ the environmental states need to be described well. Since the neuro-correlate $\mathcal{R}$ has already been shown to correlate positively with performance on the active categorical perception task [Marstaller et al., 2013], and been shown to boost performance in learning the task when the GA was augmented with $\mathcal{R}$ [Schossau et al., 2015], the same task was used here as well to allow for a direct comparison. Augmentation is done by multiplying the attained score for the task by the amount of representations the agent has about the environment (see section 3.3.11).

In this task [Beer, 1996, Beer, 2003] an agent who can move left and right has to either catch or avoid a block that is falling towards it. The agent has 4 sensors, split into two groups of 2 sensors, separated by a 2 or 3 unit space between them. Blocks of various size are dropped one at a time, with the agent having 34 time steps to catch or avoid the block. The agents in

the original experiments were evolved to catch blocks of width 2, and avoid blocks of width 4. I extended the task to try different combinations of block sizes. The blocks move 1 unit to the right or left on any given time step. On each time step, the agent receives the sensory input and can move 1 unit to the right or to the left. The agent is expected to determine the size of the block, and its direction of movement, so that it can decide whether to avoid or catch the specific block being dropped. The agents were tested over both block types and all possible permutations of starting positions relative to the sensors and block movement patterns.

Due to the configuration of the sensors and the relative location of the agent to the block, agents can only very rarely decide if a block is large or small without movement and thus need to navigate to be directly under the block - hence the "active" in active categorical perception. This can only be done if the block was observed before for at least two updates in a row, since otherwise the direction of the fall could not be determined. All this information needs to be integrated into a decision about catching or avoiding the block. It is this integration of sensor information over multiple time steps that requires agents to first evolve memory and with it representations in order to make proper decisions.

3.3.4 Number Discrimination Task

In addition to the active categorical perception task, recent work [Kirkpatrick and Hintze, 2019] has identified a different task in which I can also investigate the role of representations and smearedness in the neuro-evolutionary process. This task, referred to as the Number Discrimination Task, has agents receive two values in sequence, and require the agent to identify the larger one. This task is inspired by previous biological [Nieder, 2018, Merritt et al., 2009] and psychological [Merritt and Brannon, 2013] studies. When adapted for use *in silico*, the agents are presented with every possible pair of numbers between 0 and 5, with the values being shown as bit strings where the number of ones is the value the agent must comprehend (e.g., 00000 is 0, 01000 is 1, 11000 is 2, etc.). Agents are additionally presented

with all possible permutations of each pair (e.g., 10001 is equivalent to 01100 and both are used to represent 2). This task requires agents to store the first number and then perform a comparison to the second. The agents are first shown one value, then they are given 3 updates to process the information, followed by a second value and an additional 3 updates to process the information, before being required to make a determination of which number is larger. The world states used in calculating $\mathcal{R}$ are defined by the individual numbers used, and the information of whether the first or second value presented is larger.

3.3.5 The perception-action loop for stateful machines

Genetic programming has been used before to control robots in virtual [Reynolds, 1993, Reynolds, 1994, Handley, 1994] as well as in actual environments [Nordin and Banzhaf, 1995a, Nordin and Banzhaf, 1995b, Nordin and Banzhaf, 1997]. In order to create a meaningful controller, the evolved machine or program needs to map the sensor inputs it receives to motor outputs which control the actions of the robot controlled (see Figure 3.3). A purely reactive machine would have a direct mapping of inputs to outputs, while a stateful machine would use hidden states (memory) to store information from the sensors to use it for later decisions.

Previous approaches used genetic programming methods to evolve computer-like code running on register machine [Nordin, 1994]. Some other approaches encoded the program controlling the robot as a tree of binary operations executed on only the sensor inputs [Koza and Rice, 1992]. While it is these kinds of tree-like encodings that I seek to investigate here, this specific approach did not use hidden states and thus created purely reactive machines. Adding memory has been done differently before [Teller, 1994, Langdon, 1995, Spector and Robinson, 2002]. However, I think that the tree structure is of particular importance. This structure allows the information from all sensor and hidden states to be funneled together into a coherent representation, like all the leaves of a tree feed into the root (see Figure 3.1 for an illustration). Although intuition may indicate that this structure is correct, I must

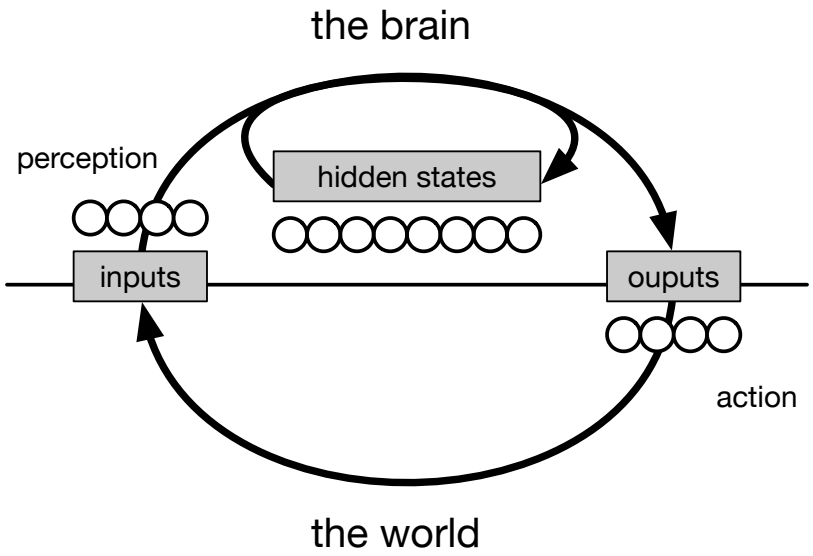

Figure 3.3: Illustration of the perception-action loop. The world is in a particular state and can be perceived by the agent. This perception defines specific input states. These states now allow the brain (computational substrate) to perform a computation, which leads to new output states. These states, in turn, cause the agent to act in the environment and thus can change the world state. In case the agent needs to remember information about the past, it has to have the ability to store these memories somewhere. Thus, any machine that needs to be stateful has to have internal (here hidden states) that can be set depending on the current input and the current hidden states. This scheme of a brain resembles the structure of a recurrent neural network, but can be generally applied to all sort of computational machines that have to be stateful and act in an environment.

empirically test the viability of these structures. This does not imply that other, for example linear programming structures or other memory models, cannot perform such functions. As such, the task here is to use genetic programming to encode tree like computational structures that can take the inputs of the machine and the hidden states into account to create new outputs while also setting hidden states in the process. A typical recurrent neural network [Russell et al., 2003] does this by extending the input and output layer of an artificial neural network and then recurring the additionally computed output states back to the inputs (see Figure 3.1 RNN). This is exactly the way how Cartesian genetic programming [Miller, 2011] would be used to create systems that can take advantage of hidden states. Similarly, Markov Brains [Hintze et al., 2017] use the same concept of recurrence to store hidden states (see Figure 3.1 Markov Brain). In fact, when Markov Brains use the same nodes that CGPs use [Hintze et al., 2019], they become very similar to each other. This kind of recurrence is very capable of creating systems that are stateful and have representations that are condensed, but again are not necessarily tree-like, and rather are more arbitrarily connected networks. Beyond using Markov Brains equipped with the computational nodes found in CGPs I use two other methods to create genetic programming trees that are stateful. GP-forests use one tree per hidden and output state and can use default values as well as inputs and hidden states from the last update[2]. GP-vector-trees are generic trees but instead of performing computations on single continuous variables, they execute vector operations. The vector supplied contains the hidden states of the last update as well as the current inputs, and part of the vector defines the output computed. I assume that this obvious extension of normal variables to vectors has certainly been done previously, but I can not find an explicit reference for this approach. Generally speaking, here I introduce one form of recurrence into genetic programming tree structures. The programming language push [Spector and Robinson, 2002] should in principle allow for the same structures to emerge. However, these structures would need to arise by chance, in comparison to the models that I introduce here which achieve the

[2]Inspired by the multiple trees used to encode memory in Langdon 1995.

desired structural goals by their definition.

3.3.6 Markov GP brains using CGP nodes

Markov Brains have been extensively on similar tasks and also to introduce and confirm the definitions of representations used here [Marstaller et al., 2013]. In a nutshell, a genome is used to encode the computational units of a Markov Brain. The genome defines the function and connectivity of these units, which can connect to input, output, and hidden states. The connections allow the computational units to read from these states and compute new outputs and new hidden states. As such a Markov Brain defines the update function of a state vector from time point t to $t+1$. Some values of this vector are inputs, some outputs, the rest hidden states. This is very similar to how an RNN is constructed. The number of hidden states in a Markov Brain or RNN can be chosen by the experimenter arbitrarily. Generally speaking, too many states slows evolution down, too few states makes evolving the task impossible. The number of hidden states used here (8) was found to be relatively optimal for the task at hand [Marstaller et al., 2013]. The computational units a Markov Brain uses were originally probabilistic and deterministic logic gates. As such, a Markov Brain defines a hidden Markov Model conditional on an input sequence [Bengio and Frasconi, 1995] - hence the name Markov Brain. For a more detailed description of Markov Brains see Hintze et al. 2017 [Hintze et al., 2017]. In this work, I use Markov Brains with CGP gates to approximate a CGP network. This creates a dynamic, GP-like structure that can be contrasted with the more rigid tree structures.

3.3.7 Genetic Encoding of GP Brains in a tree-like fashion

Genetic programming trees and also Cartesian Genetic Programming define the computations they perform by identifying computational nodes, their connections, and how they receive inputs. When evaluated, they take those inputs to perform all computations defined by their structure and return a solution. When mutated, the nodes and inputs can change, as

well as rewiring between the components can occur. In addition, when for example genetic programming trees are recombined, sub-branches of the trees are identified and exchanged between the trees [Banzhaf et al., 1998]. Since mutations directly identify the components to be changed, this could be called a *direct encoding*. An alternative to this approach are *indirect encoding* schemes which can sometimes provide an advantage during evolutionary adaptation, such as the hyper-geometric encoding for NEAT [Clune et al., 2011]. Markov Brains, for example, use a type of indirect encoding where a genome specifies genes, and these genes then define the computational components and their connectivity. Mutations happen to the genome and not on the components themselves. The genetic programming substrates used here are defined in a similar *genetic encoding* scheme. I use the term *genetic encoding* to imply that it is not entirely direct nor as indirect as other systems might be.

Here different kinds of computational structures (GP-Forest and GP-Vector Brains) are defined by such a genetic encoding scheme. Starting from a root node, new nodes have to be sequentially added. Therefore, a linear genome (vector of bytes) is scanned for start codons (a set of two numbers) which define the start of a gene. The numbers following such a start codon are then used to define the mathematical operation of a node, and how this node is inserted into the genetic programming tree (see Figure 3.4). Observe that the order of genes in the genome matters, genes found earlier encode nodes that will be inserted into a tree earlier. Mutations affect the genome and can not only shuffle genetic material around but also insert and delete whole parts. As such, genes can disappear or be created *de novo*. This means, that at the time when a node is created by reading the gene defining it, it is not known if other nodes will be appended to it later or not. Since each node in these implementations has two possible inputs, each gene encodes default values or addresses for hidden states or input values. If another node becomes added later it replaces the default value with it's own output.

Since nodes are sequentially added, they always get appended to an already existing node. To know where, each gene also encodes a continuous number $[0.0, 1.0]$ that specifies where in

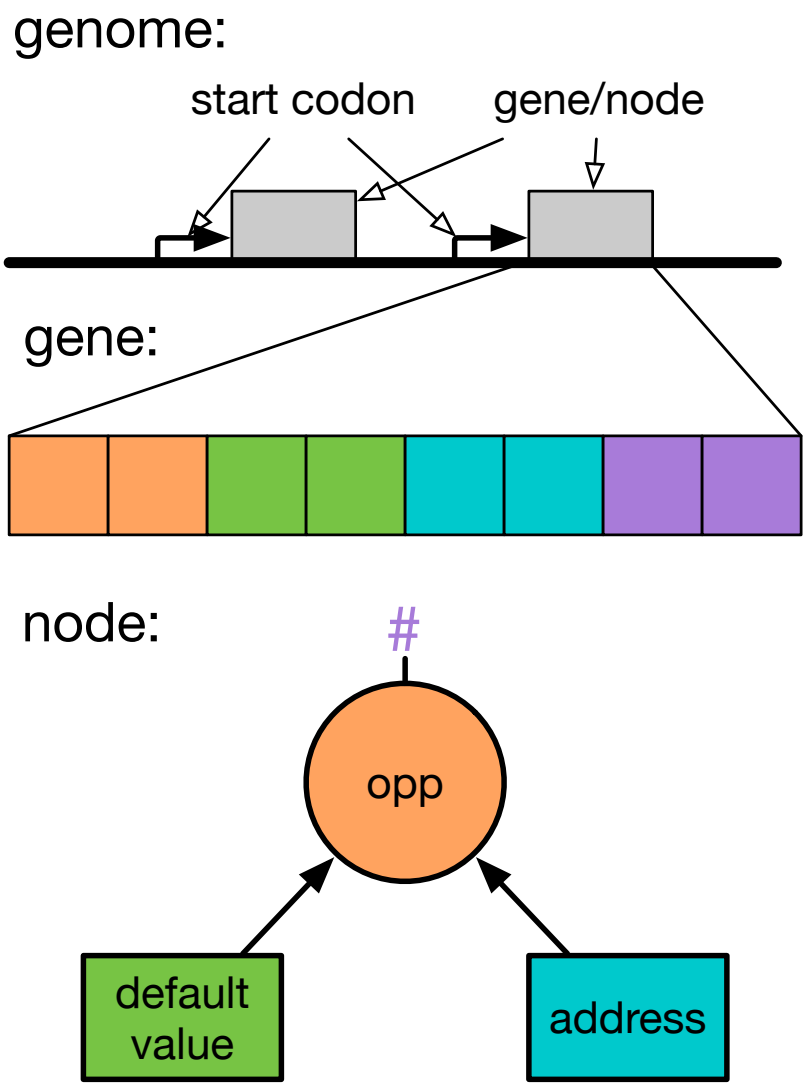

Figure 3.4: Schematic overview of the genetic encoding for computational nodes. The genome is a sequence of numbers (bytes), and specific subsets of numbers (42, 213) define start codons. The sequence of numbers behind the start codon define a gene. As such, a gene stores all information necessary to define all properties of a computational node as it is needed to function within a genetic programming tree. Specifically: the computation the node has to perform (orange), the default for its left input (green), the default for its right input (blue), and the address where it needs to be inserted (purple). Observe that the defaults can be specific values (as seen for example for the left input of the node), or the address of a hidden node as seen for example in the right input of the node.

the tree the node has to be added (see Figure 3.5 for an illustration of that process).

This scheme applies to both GP-Forest and GP-Vector. However, what each gene encodes differs with respect to the function of the node or the specific default values or addresses.

3.3.8 GP-Forest Brain

Any kind of computational system controlling an agent that needs to be stateful, cannot just map outputs to inputs but instead has to rely on hidden states which are also sometimes called recurrent states. In the case of an RNN for example, extra outputs that the network produces are just mapped back as additional inputs. Similarly, Markov Brains use inputs, outputs, hidden states upon which all computations are performed. While the inputs are generated by the environment the Markov Brain generates the outputs and the new hidden states to complete a perception-action loop. In order to allow for the same computational complexity in a genetic programming tree, I need to somehow make it recurrent. Conventional trees compute a single output based on a selection of inputs. However, this would allow us to only create systems without hidden states. Here I use, what I call a genetic programming forest (GP-Forest), where each output, as well as each hidden state, is computed by an individual genetic programming tree (hence the forest analogy). Each tree has access to all inputs and hidden states and computes new outputs and new hidden states at every update. The tree itself does not allow for cycles. However, after each update of the entire tree, the new hidden states as well as new inputs are again made available to the tree. This is what creates the recurrence, which happens at the time step boundaries. For a list of computations that each node can potentially compute see Table 3.1. The number of hidden states plus the number of output nodes to be computed defines the number of trees needed to be specified. To allow for a proper comparison of GP-forests with Markov Brains the number of hidden states used was kept the same in all brains (Markov Brain, GP-forests, and GP-Vector Brain see below).

The genetic encoding specifies for each newly added node which of the trees it should be

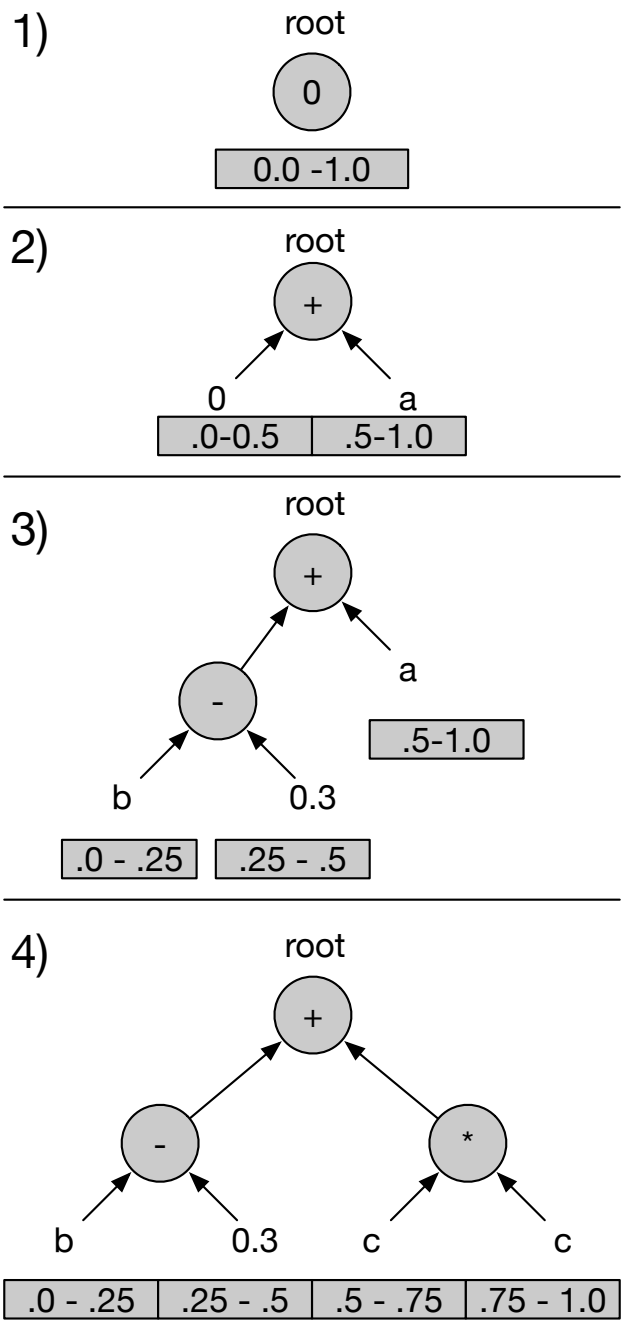

Figure 3.5: Illustration of tree growth by adding nodes. 1) in the beginning the tree is defined as an empty root, that defaults to an output (here 0.0). Once the genome is parsed and a new gene that encodes a node has been found, the node replaces the root node. The left and right inputs now define two ranges $[0.0, 0.5]$ and $]0.5, 1.0]$. 3) Each new node to be added has a specific address (a value from $[0.0, 1.0]$ as defined by the gene, see Figure 3.4 the purple component of the gene). This number defines which default input to replace. The added node itself subdivides the range it connected to, in order to allow new nodes to again be added. 4) The concept of adding nodes and subdividing the connection ranges continues until the tree is built. In this way, a new node will always have a place in the tree. However, since nodes are added sequentially, deletions of genes/nodes will affect the topology greatly, but will not make it impossible to add nodes.

Table 3.1: The possible mathematical operations of each node in a GP-Forest brain. Each node has two possible inputs: L and R, performs a computation on them, and returns the value of said computation. The last category of LOGIC consists of 16 independent binary logic operations (e.g., AND, NAND, NOR, XOR, OR, etc.) performed on the L and R input:

Operation	Return value
NOP	0.0
LEFT	L
RIGHT	R
ADD	L+R
SUB	L-R
MUL	L*R
DIV	L/R, 0.0 if R==0.0
EQU	1.0 if L==R, else 0.0
NEQU	1.0 if L!=R, else 0.0
LOW	1.0 if L¡=R, else 0.0
HIGH	1.0 if L¿=R, else 0.0
LOW	1.0 if L¡R, else 0.0
+ 16 LOGIC	all 16 possible Boolean logic operations performed on L and R

appended to as well as the location in the tree (see Figure 3.5 to see how nodes are added, see Figure 3.6 for a structural overview of a GP tree brain).

3.3.9 GP-Vector Brain

While genetic programming forests can deal with the hidden state problem by adding many trees, one for each new state to be computed, GP-Vector brains try to solve this problem differently. I assume that the input states, output states, and hidden states form a continuous vector. At each update of the brain, a new vector has to be computed. Therefore, I extend the genetic programming paradigm that conventionally allows nodes to be mathematical operators for single variables to now compute vectors. Each node is now defining a vector operation, and the inputs of each node can be constant vectors or the current state vector. Regardless, such a tree of vector operations will result in a newly computed vector which itself contains the values of the current hidden state as well as the outputs of the agent's brain. Inputs from the environment are fed into this vector before each update of the brain.

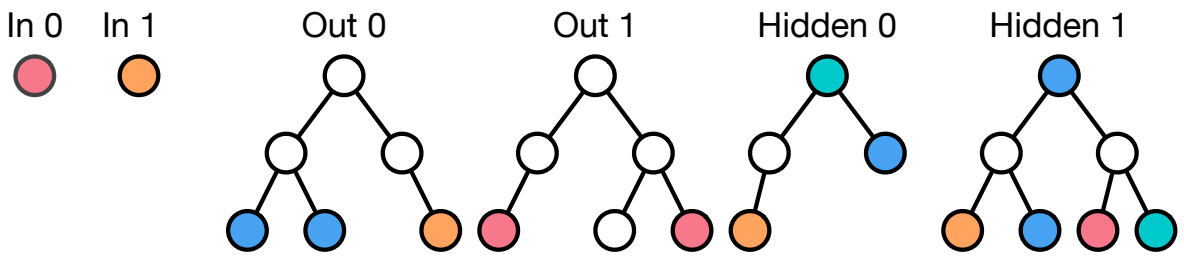

Figure 3.6: Illustration of a GP-Forest. In this example, I assume the system to have two input nodes (red and orange), two output nodes (white), and two hidden nodes (green and blue). When decoding the genome, nodes can get added to hidden and output nodes sequentially. Each node can either use a default input (white) or use the value specified by an input or hidden node. For example, the tree added to the output node 1 reads from the orange input node, and twice from the blue hidden node. When a new brain state has to be computed, all trees (outputs, hidden) are evaluated simultaneously, and consequently a new set out outputs and hidden states become defined, based on previous inputs, and hidden states.

Each node is again encoded by a gene which provides the mathematical operation to be performed, where in the tree it gets added, and it defines if the default input is the state vector, or defines an individual constant vector. Remember that when a node is added, it replaces one of the defaults of the current node with its own output (see Figure 3.7 for a structural overview of a GP-Vector brain).

Each node can now perform vector operations on its inputs and will return a vector as a result of that computation. However, such vector operations are often computationally expensive, and maybe not always necessary. Thus, nodes are also allowed to perform operations on the individual elements of the vectors. Each gate uses either the result of other nodes, a default vector of zeroes, or the state vector. These vectors are called L and $\mathcal{R}$ to capture the idea that each node in the tree can have two inputs, one from a left one from a right branch. If an addition on these vectors is performed the resulting output O is just $L + R$. Which means that every element O_i is the result of $L_i + R_i$ for all i. To allow for individual elements to be manipulated by each node, all operations are also defined as "point" operations. An $ADD_{i,j}$ would now compute an addition only on the i element of vector L and the j element of vector $\mathcal{R}$. The result would be $O_i = L_i + R_j$ while all other elements of O would be those defined by L.

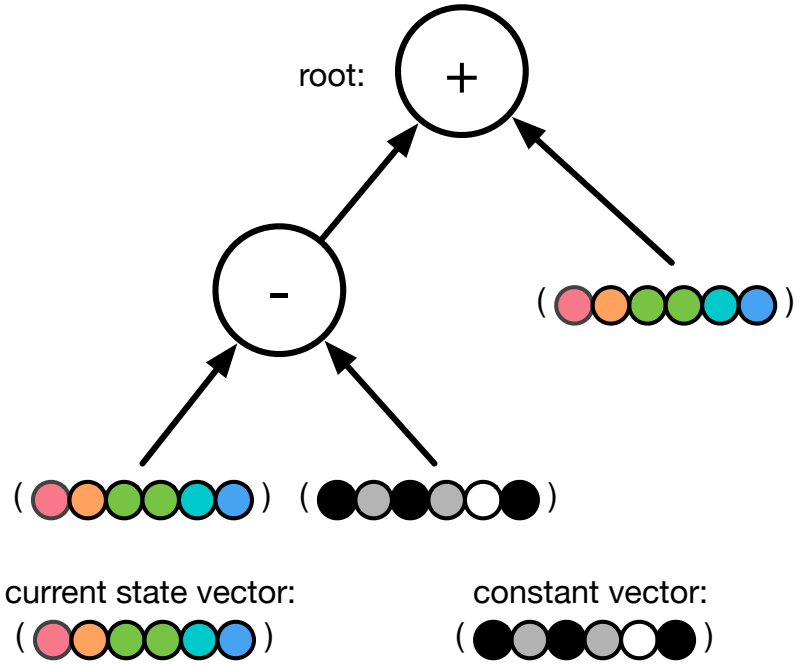

Figure 3.7: Illustration of a GP-Vector Brain. This brain is structurally identical to conventional genetic programming trees, except that it can execute computations on vectors instead of single values. The inputs to nodes can be the state vector, including all inputs (red and orange) and all hidden states (blue to green). Alternatively, nodes can use default vectors as inputs. These defaults are defined by the genes encoding each node.

Table 3.2: The allowed mathematical operation for the nodes of a GP-vector-brain. Observe that all operations are either performed on all elements of the L and $\mathcal{R}$ vector resulting in an output vector O, or on individual elements of L and $\mathcal{R}$. Only the LOGIC operation is only defined as "point" operation and not on the entire vector.

Operation	Return value
ADD	$O = L + R$
SUB	$O = L - R$
MUL	$O = L * R$
DIV	$O = L/R$
SIN	$O = \sin L$
COS	$O = \cos L$
TANH	$O = \tanh L$
MIN	$O = \min L$
MAX	$O = \max L$
MEAN	$O = 0.5(L + R)$
ABS	$O =\mid L \mid$
NEG	$O =\mid -L \mid$
LOGIC	$O_i = \text{logic}(L_i, R_j)$

The size of the vector is defined by the number of inputs, outputs, and hidden states. While the number of input and output states is defined by the task, the number of hidden states for this brain was set to be equal to the number of hidden states in the Markov Brain, to allow all brains to be directly comparable.

3.3.10 Evolutionary Process

Each of the agent populations with different brain types was evolved for 40,000 generations in populations of 100 organisms. The start population was initialized with random genomes of length 5000. The genome size was capped at a maximum of 20,000 sites and a minimum of 2,000 sites. For each experimental condition, I ran 400 independent replicate experiments. At the end of the evolutionary process, the line of descent was reconstructed, and all analyses were performed on these agents. All experiments, using each of the three agent types and two worlds, were implemented using the MABE code framework [Bohm et al., 2017] which among many other things includes an extensive version of Markov Brains as well as all other tools required to perform neuro-evolution on arbitrary substrates and tasks.

In both worlds, I used the number of correct decisions C (e.g., blocks caught or avoided correctly, pairs of numbers correctly ordered) and incorrect decisions I (e.g., blocks caught that should have been avoided, pairs of numbers incorrectly ordered) to define the fitness function as:

$$W = 1.1^{(C-I)} \tag{3.5}$$

This function is an exponential fitness function where an additional correct decision of the agent does not linearly but exponentially improves fitness. This increases the chances of an agent to be selected for reproduction exponentially, which in my experience allows the GA to select for functional agents better. The exponential base value (1.1) was taken from previous work [Marstaller et al., 2013].

After the fitness has been computed for every organism in the population, I use roulette wheel selection to determine which organisms contribute offspring to the next generation [Thomas, 1996]. Roulette wheel selection is a method that ensures selection to be directly proportional to fitness. Every time an agent made an offspring the genome experienced gene duplications (with probability $2*10^{-5}$ per nucleotide of the parent genome, duplicate the next 128 to 512 contiguous sites), deletions (with probability $2*10^{-5}$ per nucleotide of the parent genome, delete the next 128 to 512 contiguous sites), and point mutations (with probability $5*10^{-3}$ per nucleotide of the parent genome, randomize the site).

3.3.11 Augmenting with R

In order to augment the performance of the GA, not only the performance of each agent is measured but also the amount of representations each agent has about its environment. In the block catching task specifically the size of the block, the direction of it, and whether it is to the left or right. In the number comparison task, representations are about the individual values of the numbers to be compared, and the ordering of the pair (i.e., if the first or the second was smaller).

To assess performance the number of correct catches (C) and misses (I), or the number of correct (C) versus incorrect answers (I) are counted. $\mathcal{R}$ is quantified as described above, and $\mathcal{R}_{max}$ is the maximum hypothetical value for $\mathcal{R}$, which is dependent on the environmental and memory states. The total fitness of the agent is determined by the following equation:

$$W = 1.1^{(C-I)}(1 + \frac{\mathcal{R}}{\mathcal{R}_{max}}) \tag{3.6}$$

Alternatively I could have used a multi-objective optimization method [Deb, 2001] that would allow some agents in the population to have a high score and low $\mathcal{R}$, while others might perform poorly but have proper representations. This keeps the diversity high, but only a recombination step might take full advantage of such diversity. Recombination however,

would introduce another level of complexity that might have confounded the results. As such, I did not pursue this option here.

3.4 Results

3.4.1 GP trees evolve to have representations

Each system responds differently to optimization by a genetic algorithm (GA), and as such I expect GP-Vector, GP-Forest, and Markov Brains using GP gates (further called MBwGP) to generally evolve to solve the task at hand but be optimized at different speeds. Similarly, systems might tend to get stuck in local optima differently, and thus, the resulting performance might differ between brains.

As expected, I find the different brains to respond differently to the optimization of the GA (see Figure 3.8). I find all systems to evolve generally well, however, GP-Forests tend to perform better on the number task (see Figure 3.8 right panel). MBwGP and GP-Forest are almost indistinguishable with respect to their performance in the active categorical perception (APC) task (see Figure 3.8 left panel). GP-Vector brains, however, perform poorest on both tasks.

Interestingly, I find a clearer delineation of result with respect to the degree in which these systems evolve the ability to have representations (see Figure 3.9). GP-Forests evolve the highest degree of representations, followed by MBwGP, trailed by GP-Vector Brains. This is particularly interesting since previous work [Marstaller et al., 2013] found that RNNs evolve representations early, but struggle with improving their performance, even though they already have a high level of representations. Markov Brains on the other hand evolve representations slowly over time, and thus can only improve their performance later, after they have sufficient representations. GP-Forest Brains seem to evolve like Markov Brains but store more representations.

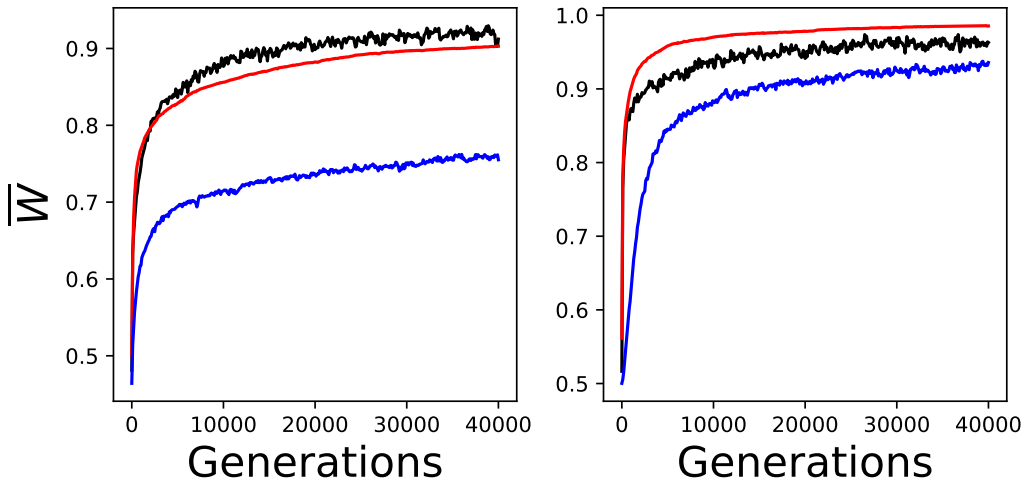

Figure 3.8: Performance over time for all 3 brain types (GP-Forest = Black, MBwGP = Red, GP-Vector = Blue). Left plot is active categorical perception, right is number comparison

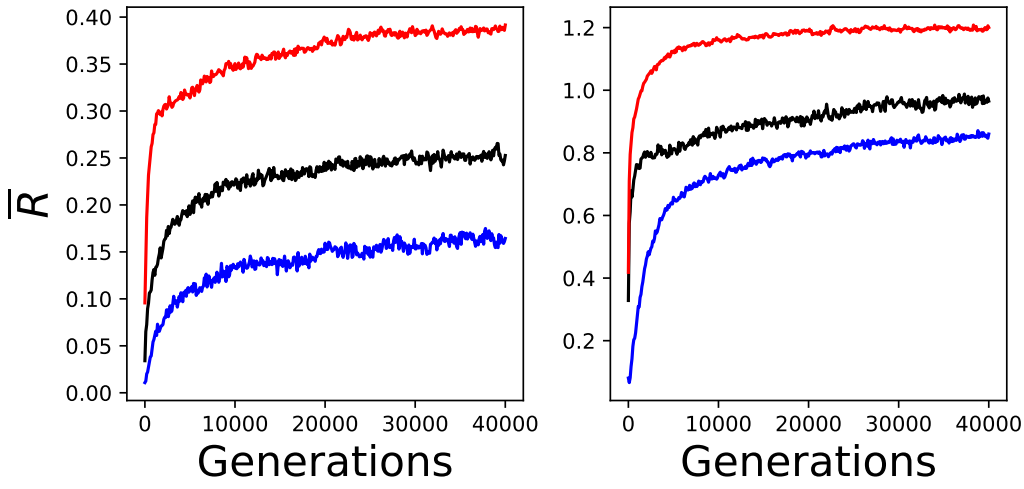

Figure 3.9: $\mathcal{R}$ over time for all 3 brain types (GP-Forest = Black, MBwGP = Red, GP-Vector = Blue). Left plot is active categorical perception, right is number comparison

3.4.2 Does Augmentation using $\mathcal{R}$ improve the performance of a GA?

I know that one can augment the optimization a GA performs by using $\mathcal{R}$ (see Augmenting with $\mathcal{R}$). The idea is, that mutations that improve an agents ability to store more representation would normally be neutral as they not necessarily also immediately provide an advantage to performance. However, in order to improve performance, often representations are needed. The augmentation process, therefore, turns mutations that improve $\mathcal{R}$ but are neutral into beneficial ones. While this is true for Markov Brains and RNNs, it is not clear is the GP-forest or GP-vector brains will behave the same.

I tested this hypothesis by evolving all three types of brains again on both environments, but also measured the representations they have during evolution, and used this quantity to augment the GA. This is in contrast to the results above, which merely examined the unaugmented GA. I find that this augmentation has almost no effect on GP-Forests and only little on MBwGP brains (see Figure 3.10). However, it significantly improves the performance of GP-Vector Brains in the active categorical perception task. I also find that GP-Forests and MBwGP already evolve the ability to store representations well, but GP-Vector Brains struggle. This supports the hypothesis that augmenting a GA by using $\mathcal{R}$ works only for systems that struggle to form representations in the first place.

I further find that the augmentation with $\mathcal{R}$ might not improve performance, but still had an effect on the ability to store representations in each system. Each brain when evolved using $\mathcal{R}$-augmentation evolved to have more representation over evolutionary time than the unaugmented control (see Figure 3.11). This proves that the augmentation indeed has an effect, but again, only improves the performance of the GA when a system struggled to evolve representations.

The most notable finding is that GP-Forest Brains do evolve a large degree of representa-

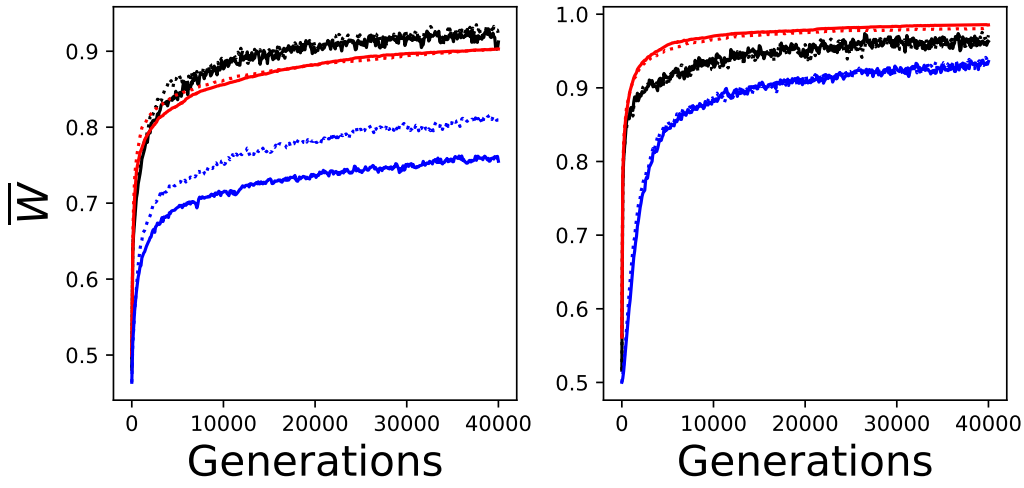

Figure 3.10: Performance over time for all 3 brain types (GP-Forest = Black, MBwGP = Red, GP-Vector = Blue). Left plot is active categorical perception, right is number comparison. Solid line is control, dotted line is augmented with $\mathcal{R}$

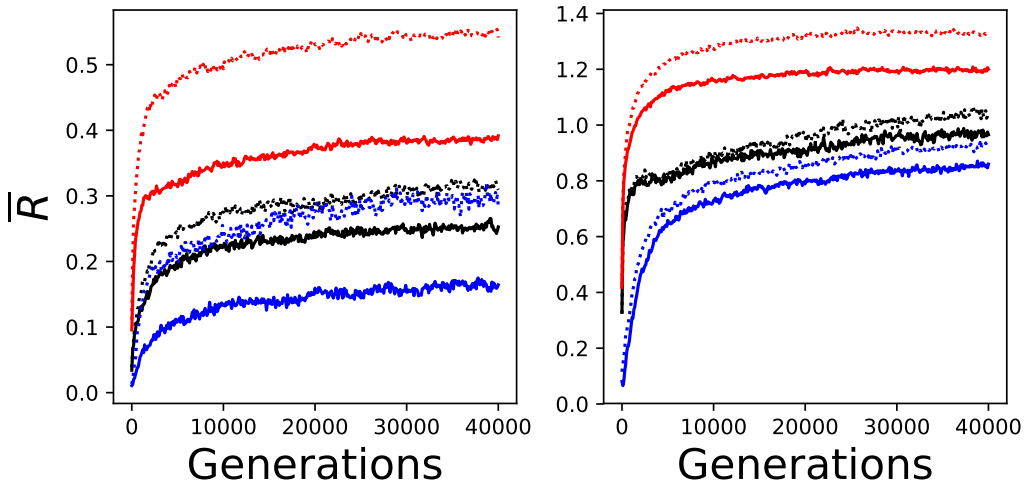

Figure 3.11: $\mathcal{R}$ over time for all 3 brain types (GP-Forest = Black, MBwGP = Red, GP-Vector = Blue). Left plot is active categorical perception, right is number discrimination. Solid line is control, dotted line is augmented with $\mathcal{R}$

tions, and respond to the augmentation with an increase in $\mathcal{R}$ but do not significantly benefit from the augmentation with respect to their performance.

3.4.3 Smeared Representations

I found earlier [Hintze et al., 2018] that brains who evolve to have sparse and condensed representation are more robust to noise, and conjecture that overly smeared representations negatively affect performance and evolvability. When representations are smeared, every mutation changing the system would affect the computation of all concepts. A similar argument has been made explaining the benefit of the dropout method in deep learning [Srivastava et al., 2014]. By removing nodes while training an ANN, representations might become more condensed, improving the ANNs ability to generalize. Another way of thinking about this is that the more clearly separated the concepts in the cognitive system are, the better it can classify. I already showed that neuro-evolution of RNNs and LSTMs results in systems that smear representations greatly, while Markov Brains evolve to have sparse condensed representations [Hintze et al., 2018]. The question here is, if the structures of genetic programming trees helps to further condense representations, or if these systems smear representations never the less?

I find that the degree to which these systems evolve smeared representations again varies greatly between the systems. While MBwGP and GP-Vector Brains only smear their representations marginally when evolved to solve the active categorical perception task, GP-Forest Brains smear their representations much more (see Figure 3.12 left panel). When evolve to solve the number task, both GP-Forest and GP-Vector smear more than the MBwGP (see Figure 3.12 right panel).

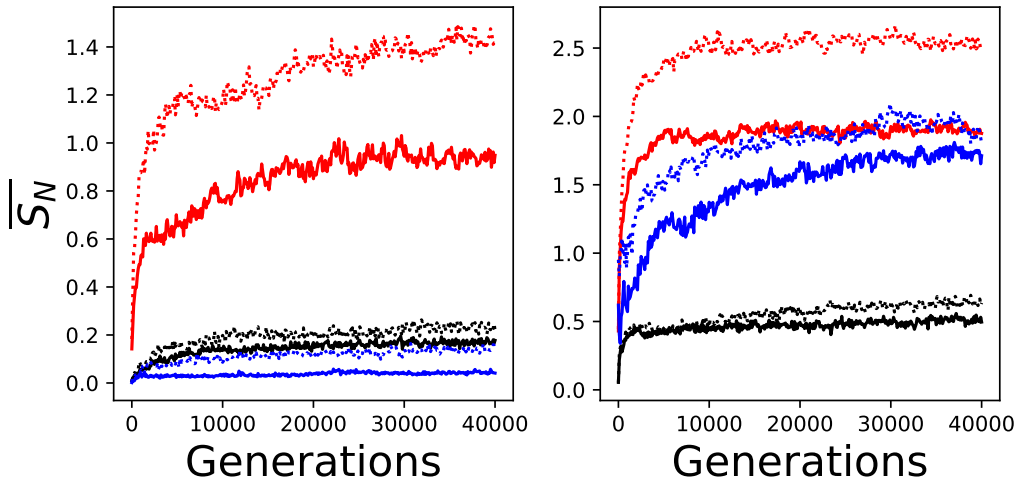

Figure 3.12: Smearedness of Nodes over time for all 3 brain types (GP-Forest = Black, MBwGP = Red, GP-Vector = Blue). Left plot is active categorical perception, right is number comparison. Solid line is control, dotted line is augmented with $\mathcal{R}$

3.5 Discussion

I used neuro-evolution to optimize the performance of three different computational systems. Markov Brains that use GP nodes are similar to CGPs and evolve a network topology that I know is capable of forming representations which are not smeared. I confirmed that Markov Brains using GP nodes benefit from augmenting the GA using $\mathcal{R}$, and asked if the same principles apply to other cognitive topologies. Of particular interest are the tree-like structures from genetic programming trees. Other than the topology of ANNs which connects every node with every other node and leads to extreme smearing of representations [Hintze et al., 2018] I assumed that tree structures would behave differently in this regard. The tree structure can take advantage of all possible inputs and hidden states but converges them into a single output at the end. The assumption was, that these systems would condense representations better and thus smear less. The hope was that such a construction allows a system to take full advantage of having a high degree of representations while at the same time not smearing them.

While the GP-Vector implementation performed poorly compared to GP-Forests and

MBwGP, I found that GP-Forests evolve equally well or even better when compared to MBwGP (see Figure 3.8). GP-Forests also evolve to have more representations than the less structured MBwGP Brains (see Figure 3.9), and these representations do smear (see Figure 3.12). However, augmenting the GA with $\mathcal{R}$ has no effect on their performance (see Figure 3.10) and only increases the amount of representations they evolve to have as expected (see Figure 3.11). As such, GP-Forests have the ability to evolve representations and take advantage of them by construction, and are thus "immune" to further augmentation by using $\mathcal{R}$.

Lastly, the smearing of representation in GP-Forests is much less than it has been observed in RNNs before [Hintze et al., 2018]. This further supports the idea that the tree structure provides a benefit over either the networks that Markov Brains or CGPs form and over the fully connected regular structures that RNNs have when it comes to having representations and taking advantage of them over the course of evolution.

In other words, GP-Forest Brains evolve equally well or better than Markov Brains and form representations easily by their construction, and thus do not benefit from further augmentation with $\mathcal{R}$. At the same time, they do smear their representation but not to the degree that it becomes a problem as it does in RNNs.

3.6 Conclusions

The two genetic programming systems (GP-Forest and GP-Vector) solve the quest to make genetic programming trees have recurrence, thus allowing them to become stateful machines. In the case of GP-Forests, their structure provides an advantage over other regular (ANN) or network structures (Markov Brains) with respect to evolving meaningful representations. The immunity of GP-Forests to the augmentation with $\mathcal{R}$ methods shows that this system automatically retains representations sufficiently well, without smearing them, by nature of its structure.

This insight suggests that one should take further advantage of tree-like structures in neuro-evolution. The capabilities of the GP-Forests also suggests investigating tree structures and how they affect the concept of smearedness in the context of deep learning.

3.7 Acknowledgements

I thank Stephan Winkler for insightful discussions on hidden states in genetic programming trees, and for implementing the prototype for GP-Forests in EvoSphere.

Chapter 4

Evolutionary Dynamics Effects Account for the Improvement Caused By $\mathcal{R}$-Augmentation

A portion of this chapter was originally published as "Kirkpatrick, D., & Hintze, A. (2020, November). Evolutionary Dynamics Effects Account for the Improvement Caused by R-Augmentation. In 2020 7th International Conference on Soft Computing & Machine Intelligence (ISCMI) (pp. 96-100). IEEE." A number revisions and additions have been made for inclusion in this dissertation.

4.1 Abstract

Previous work has found a method for augmenting a genetic algorithm (GA), referred to as $\mathcal{R}$-augmentation, that produces better-scoring results earlier in evolutionary time. $\mathcal{R}$-augmentation works by using an information-theoretic measure quantifying mental representations (denoted as R) as an additional contribution to the fitness function of the GA. It has been shown that this method improves the performance of the GA by encouraging

the evolution of high-performing artificial agents controlled by Markov Brains or Recurrent Neural Networks. Different mechanisms could explain this phenomenon, ranging from structural changes to differences in the associated mutational effects. Here I examine a range of potential sources of improvement and demonstrate that the majority of the improvement is caused by $\mathcal{R}$-augmentation altering the evolutionary dynamics of the GA, reflected in the changed effects of mutational operators and changes to the fitness landscape.

4.2 Introduction

Evolutionary algorithms used to train artificial intelligences can produce novel and often unexpected results as compared to other types of optimization, particularly reinforcement learning [Lehman et al., 2018]. However, an evolutionary optimization system may take a long time to reach an optimal solution to a problem, if it ever reaches that optimal solution at all. Many methods have been proposed to reduce the run time of these evolutionary algorithms, in order to create better performing solutions with less compute time. Previous work has identified one method of boosting the evolutionary process, referred to as $\mathcal{R}$-augmentation, that improves GAs by promoting the evolution of artificial intelligences with increased mental representations [Schossau et al., 2015]. This augmentation process has been found to result in a higher number of perfect performers evolved as well as higher task performance in fewer generations, when compared to an unaugmented GA. Here, mental representations are measured as the amount of sensor-independent information that the artificial intelligence has integrated into its internal memory from the environment and are evaluated using the information theoretic neurocorrelate $\mathcal{R}$ [Marstaller et al., 2013]. Colloquially, $\mathcal{R}$ quantifies how much a system knows about its environment without looking at the environment. This method can only be applied to tasks that require agents to form recurrent internal models about their environment, which are generally stored in memory through some form of network feedback loop. $\mathcal{R}$-augmentation works by changing the original fitness function so that $\mathcal{R}$

is evaluated in addition to the task-related components (see subsection 4.3.3 for a fuller description of the $\mathcal{R}$-augmentation process). While $\mathcal{R}$-augmentation as well as augmentation using other neurocorrelates allow GAs to generate better-performing solutions in less evolutionary time, it is not clear how the improvement is generated. Although the benefit of $\mathcal{R}$-augmentation comes at a cost, previous work was unable to identify a cost-benefit trade-off beyond the computational cost of calculating $\mathcal{R}$ (see Ch. 2). Understanding how and why these augmentation methods improve the performance of GAs is a crucial step towards accelerating evolutionary search as a whole.

A number of changes to the networks created by $\mathcal{R}$-augmented GAs could account for the difference in performance as opposed to those created by an unaugmented GA. First and foremost are changes to the internal structure of the cognitive network, particularly those geared at improving access to memory. In order for representations to form, information must be maintained in the network. In order for information to be maintained in the network, it must be repeatedly propagated into a usable location, and thus form a part of an internal looping structure. Since the formation of representations requires an internal looping structure to maintain information, I will attempt to quantify the recurrent structures of the network by analysing the connections between nodes. I primarily consider two cases; first, a node may have an output that feeds back (through computational components) into its own inputs and form a self-recurrence. Secondly, a node's output may feed into another node, which is then fed back into that node, creating a two-step recurrence. Although there may be larger, more complex chains of recurrence, for the context of this paper I only consider the first two cases as a difference in the recurrent structure should be obvious there. The cognitive network that I study, the Markov Brain, is highly composable, that is it can be made of different internal computational components, referred to as gates. These gates can have internal mechanisms that depend on deterministic or probabilistic logic, or even mimic the functionality of other network structures like artificial neural networks (ANNs) or Genetic Programming (GP) trees. Moreover, recent work [Hintze et al., 2019] suggests a new paradigm

for approaching the question of how $\mathcal{R}$-augmentation works by offering a new method of configuring Markov Brains, referred to as the "Buffet Brain." [Hintze et al., 2019] suggests that evolution proceeds more effectively if the Markov Brain has access to all gate types. Here, I can allow the GA access to all gate types, and see what differences arise in the gates chosen by $\mathcal{R}$-augmented and unaugmented GAs. Presumably, if the computational substrate matters for the $\mathcal{R}$-augmentation process to be effective, I would see a difference in gate type distributions for the runs using the buffet method.

Beyond the changes to the network structure or composition, I must examine alternate sources of the difference in performance. I will frame this difference by first considering an analogous method to $\mathcal{R}$-augmentation, namely multiple objective optimization [Zhou et al., 2011]. As opposed to $\mathcal{R}$-augmentation, objectives in multi-objective optimization usually involve a trade-off, that is, they are often antagonistically linked. In $\mathcal{R}$-augmentation the relation between performance (of fitness W) and $\mathcal{R}$ is not necessarily antagonistic. Their relation could be synergistic, where high levels of $\mathcal{R}$ drive high performance. Alternatively, one might be a necessary condition for the other. For example, the amount of information present in the system could be limiting the performance, and only once the more information is acquired can performance follow. With $\mathcal{R}$-augmentation, a positive change to $\mathcal{R}$ would increase fitness and thus not be neutral. As a consequence, later mutations would then be able to take advantage of the increase in $\mathcal{R}$ and improve performance directly. Without $\mathcal{R}$-augmentation, the first mutation would be neutral and effectively invisible, and any change in performance would require an additional change in $\mathcal{R}$ through a separate mutation.

An alternative answer to the question of how $\mathcal{R}$-augmentation works involves the fitness landscape itself. The fitness landscape metaphor assumes a space defined by all possible genotypes. In this space, the neighborhood relations are defined by the mutational operators. Two genotypes that are one mutation apart would be direct neighbors, whereas genotypes that are separated by many mutations would be far from each other in the fitness landscape. The "height" at each genetic location is given by the fitness of each genotype. Without $\mathcal{R}$-

augmentation this landscape has a particular slope and ruggedness that determines the rate of adaptation. $\mathcal{R}$-augmentation changes the landscape, perhaps making it smoother than the unaugmented GA's landscape. As a consequence, the ability to cross valleys in this altered landscape might be increased, or perhaps valley-crossings are eliminated altogether. Here I examine differences in the network structure and composition as well as several different indicators of a change to the evolutionary dynamics of the augmented and unaugmented GAs to gain a fuller understanding of how $\mathcal{R}$-augmentation results in increased task performance.

4.3 Materials and Methods

4.3.1 Markov Brains

Markov Brains are a form of neural network composed of computational modules (most often Boolean logic gates) that read and write from data nodes. The entire topology of the network can evolve along with the functionality of each individual computational module that connects nodes. The topology and functionality of the network is encoded in a genome that encodes an arbitrary number of genes, where each gene encodes a portion of a computational unit. The genes are delimited by markers ("start codons") that specify a gate, with subsequent genes determining the gate's function and connectivity. Gates can read from and write into common data nodes, allowing for data interaction and complex computations. Data nodes can receive sensor information and provide output to generate motor actions, and additionally read from and write to a persistent set of recurrent nodes that allow for memory and representations. The sensor input and motor output make it possible for the brain to behave as an agent in a virtual environment. In principle, Markov Brains can use almost any conceivable input-output computational structure as a gate, however here I limit the gate types to deterministic logic, probabilistic logic, Genetic Programming (GP) logic, artificial neural network (ANN), and Neural Augmenting of Evolved Topologies (NEAT) derived mathematical functions. In contrast, the original $\mathcal{R}$-augmentation only allowed deterministic logic gates. I use a version

of Markov Brain from recent literature [Hintze et al., 2019] - referred to as a Buffet Brain - which allows the Markov Brain to select from all of the listed gate types at once. For a fuller description of Markov Brains, see [Hintze et al., 2017].

4.3.2 Active Categorical Perception Task

Agents are evaluated in an active categorical perception task which has been extensively described in prior literature [Schossau et al., 2015, Kirkpatrick and Hintze, 2019]. In brief, agents can move laterally and have to either catch or avoid blocks of different sizes dropped at them. Due to a blind spot in the agent's sensor, a single observation is not sufficient to unambiguously determine either the size of the block or the direction in which the block is moving. Thus, the agents have to actively move in order to position themselves properly and then make multiple observations to categorize block size and movement direction. This information has to be stored so that the agent can use it to successfully catch the small blocks or avoid the large blocks at a later time point. An agent's ability to complete the task is tested over 64 different drop conditions - permutations of large or small block size, block dropping to the left or right, and 16 different block start positions relative to the agent. The performance of the agent in the task is measured terms of in the number of correct and incorrect decisions made over all drop conditions.

4.3.3 Representations

"Representation" is a term I use to quantify the information about the world, environment, or task stored internally by the network that is not shared with the current sensor input. In other words representations are the stored information in the brain that correlates with information in the environment but does not correlate with sensory clues. Such a quantity can be measured using the information-theoretic measure $\mathcal{R}$ [Marstaller et al., 2013], which is used in the process of $\mathcal{R}$-augmentation to change the fitness function so that mental representations are evaluated along with task performance.

Generally speaking, information is shared between the world, the sensors, and the hidden states of the computational system. Each state can be expressed as an independent random variable, and their intersection defines how much each of these random variables can explain the other (see Figure 4.1). The representation $\mathcal{R}$ (how much the memory states M know about the environment E given the sensory data S) can be expressed as:

$$\mathcal{R} = H(E : M|S) = I(E : M) - I(E : M : S) \; . \tag{4.1}$$

Here, $H(E : M)$ (from Eq. 4.1) refers to the shared Shannon entropy between world states and brain states, while $I(E : M : S)$ stands for the information shared between the environment, memory, and sensors. In practice I use the simplified equation (Eq. 3.3) found in Chapter 3.

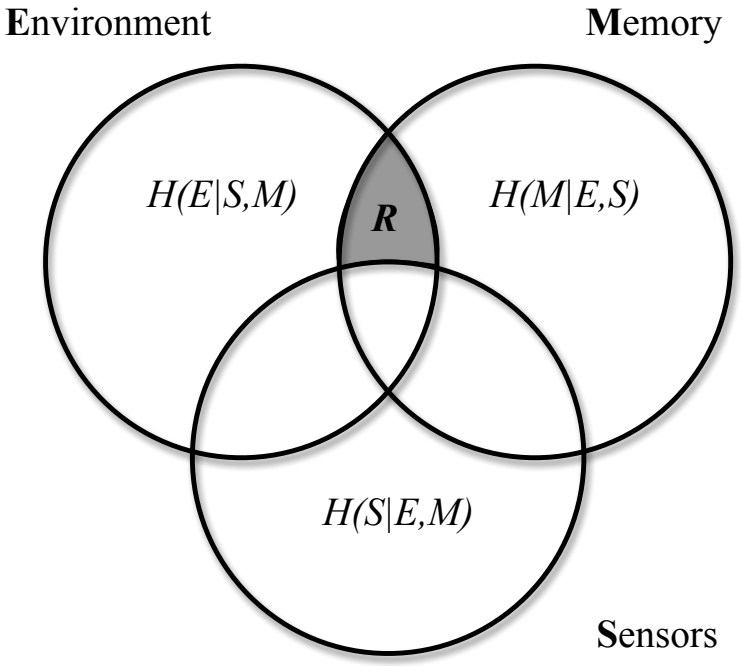

Figure 4.1: Venn diagram of entropies and informations for the three random variables E, S, and M, describing the environment, sensor, and agent internal (memory) states. The information-theoretic measure of representations $\mathcal{R} = H(E{:}M|S)$ is shaded. ©2020 IEEE.

4.3.4 Fitness Function and $\mathcal{R}$-Augmentation

To implement $\mathcal{R}$-augmentation , I supplement the standard fitness function, Equation 4.2 defined in terms of correct (C) and incorrect (I) decisions,

$$W = 1.10^{(C-I)} \tag{4.2}$$

with a term involving $\mathcal{R}$ (see [Schossau et al., 2015] or Chapter 2). The measured value of $\mathcal{R}$ is normalized by dividing by the theoretical maximal attainable value of $\mathcal{R}$, $\mathcal{R}_{max}$, which is dependent on the number of bits of information present in the specific task states and brain hidden states (here, 4 and 8 bits, respectively). The addition of this term results in a new function for fitness (W) as seen in equation 4.3 that is used in the augmented GA.

$$W = 1.10^{(C-I)}(1 + \frac{\mathcal{R}}{\mathcal{R}_{max}}) \tag{4.3}$$

The base value of 1.10 is a scaling factor that allows us to transform an otherwise linear fitness function into an exponential. Previous work [Schossau et al., 2015] has shown that such a scaling function works well, and I adopt it here.

4.3.5 Evolutionary and Experimental Conditions

I evolve agents controlled by Markov Brains [Hintze et al., 2017] in populations of 100, with 500 replicate runs for each experimental condition (unaugmented and $\mathcal{R}$-augmented GAs evolving the Buffet Markov Brain). At the beginning of each evolutionary run, I generate all agents stochastically using a replicate-specific random number seed. Over the course of evolution, agents can experience point mutations (with probability p=0.005), or insertions and deletions (with probability p=0.0002) of genetic material. At every generation, I evaluate the agents on the active categorical perception task, with both task performance and $\mathcal{R}$ calculated.

After 10,000 generations the line of descent [Lenski et al., 2003] was reconstructed, with data stored for every time point. The line of descent is the lineage (parent, grandparent, great-grandparent, etc.) of the highest-performing individual in the final population (i.e., from generation 10,000). I run two experimental conditions, the first where I use the original fitness function (Equation 4.2), and one where I apply the $\mathcal{R}$-augmentation process using the augmented fitness function (Equation 4.3).

4.3.6 Network Structure and Composition Analysis

To examine how network structures affect representations I look specifically at the connections that feed back into themselves, which is necessary for information storage and representations. Without a feedback structure, the information will not be maintained in the network, which makes memory, and therefore representations, impossible. For Markov Brains in particular, as the topology is evolved and not predetermined, these network loops must be evolved more or less from scratch, which makes their presence worth examining. To determine the looping structures, I first process each computational gate from every individual brain on the line of descent, determining in each brain which data nodes are read from and written into. With these links, I create a connectivity matrix for each brain, and determine the number of 1-cycles (where data is read from and written back into the same node) and 2-cycles (where data is read from a node, written to a second node, and then read from the second node and written back to the original node). In order for representations to be persistent and therefore useful, they must be stored in these feedback network cycles.

While studying differences in network composition to explain how $\mathcal{R}$-augmentation improves task performance, I can also examine what gate types are preferred by brains evolved by GAs with and without $\mathcal{R}$-augmentation. To study differences in the types of gates selected for, I can visualize the distribution of different gate types over evolutionary time. Furthermore, I can examine the total number of gates evolved, to get an approximation of the networks' complexity.

4.3.7 Evolutionary Dynamics Analysis

After I reconstruct the lines of descent, I compare the differences from one time point to the next. This makes it possible to calculate the change in fitness (ΔW) and the change in $\mathcal{R}$ ($\Delta\mathcal{R}$) associated with each mutation along the line of descent. While a majority of the mutations are neutral (having no ΔW and no $\Delta\mathcal{R}$) a significant number of these mutations have changes that result in a non-zero ΔW or $\Delta\mathcal{R}$. By examining how these changes vary in each experimental condition, I can unfold how $\mathcal{R}$-augmentation affects the evolutionary dynamics.

4.4 Results and Discussion

Since I am using a different selection of computational gates for the Markov Brain structure, I first replicate the results from [Schossau et al., 2015], demonstrating the utility of $\mathcal{R}$-augmentation. I find that, despite the difference in choice of gates, $\mathcal{R}$-augmentation allows the GA to evolve solutions that have higher task performance overall, and arrive at these high-performing solutions earlier in evolutionary time relative to unaugmented GAs (see Figure 4.2).

Many possible factors could explain how $\mathcal{R}$-augmentation improves performance of models created by the GA: structural changes to the network, different exploratory behavior of evolved agents, selection of different computational primitives, the order of steps in which evolution optimizes the behavior, to name only a few. In this chapter, I will examine the structural differences, component differences, and differences to evolutionary dynamics. First, I look at the gate distribution of the evolved Markov Brains. At first glance, the distribution of gates as a proportion of all gates used, as seen in Figure 4.3, does not show a noticeable difference between the Markov Brains created by $\mathcal{R}$-augmentation and those created without.

However, when the distribution is not taken as a proportion of the total gates, but rather

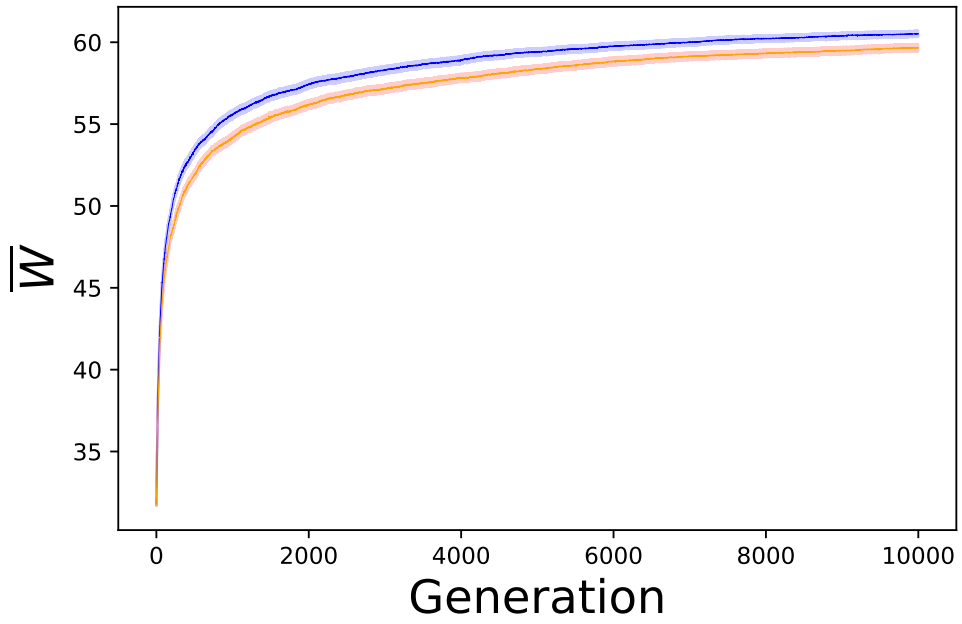

Figure 4.2: Task Performance in Buffet Markov Brains produced by Augmented and Unaugmented GAs. The blue line represents results from the lines of descent of GAs using $\mathcal{R}$-augmentation, while the orange line comes from unagumented GAs. Each line is the average performance over 500 replicate experiments. The shaded area around each line represents the standard error. $\mathcal{R}$-Augmentated GAs produce Markov Brains that have a significantly higher average performance than the Markov Brains produced by unaugmented GAs.©2020 IEEE.

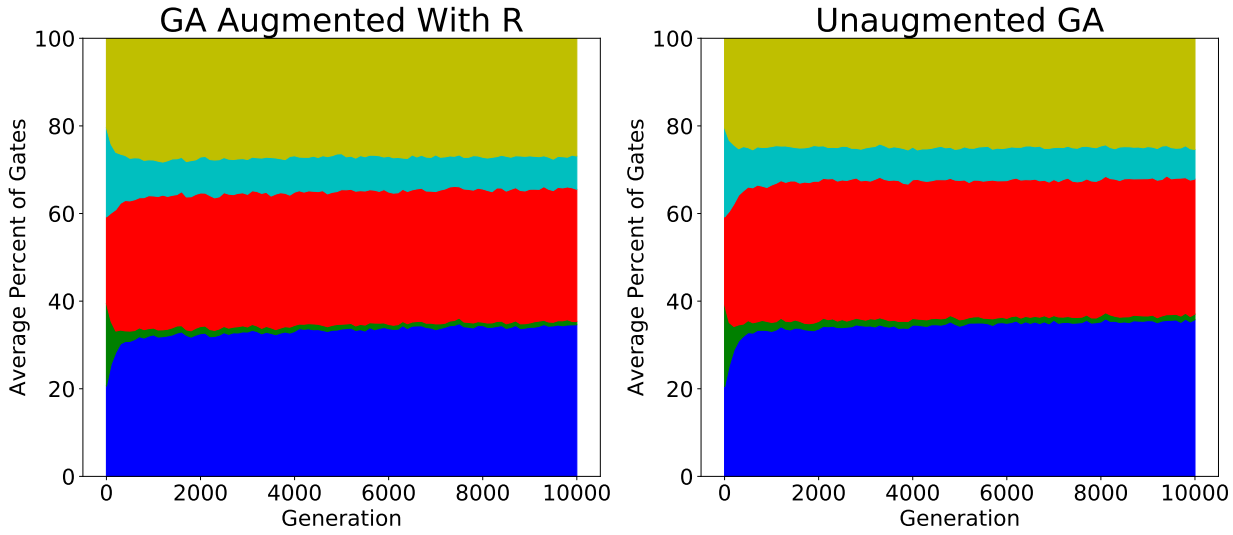

Figure 4.3: Average Proportion of Different Gate types in Buffet Markov Brains produced by Augmented and Unaugmented GAs over evolutionary time. Deterministic gates are blue. Probabilistic gates are green. ANN gates are red. NEAT gates are cyan (light blue). GP gates are yellow.

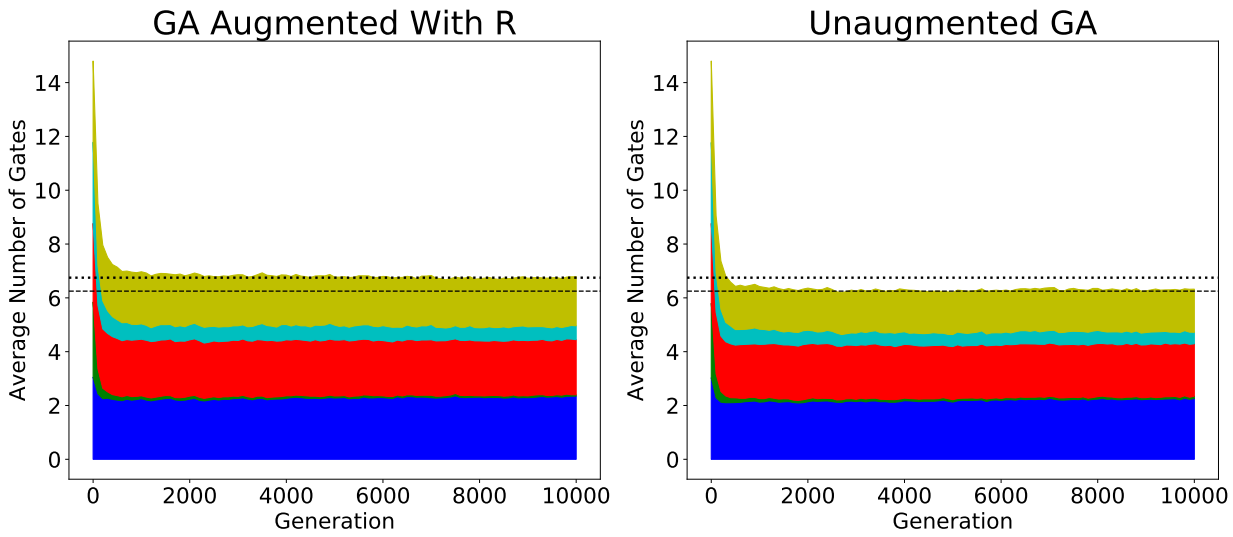

Figure 4.4: Average Number of Different Gate types in Buffet Markov Brains produced by Augmented and Unaugmented GAs over evolutionary time. Deterministic gates are blue. Probabilistic gates are green. ANN gates are red. NEAT gates are cyan (light blue). GP gates are yellow.

the average count of gates, as shown in Figure 4.4, a more interesting effect becomes clear. The Markov Brains created using $\mathcal{R}$-augmented GAs have a higher average number of gates, where the difference is approximately half a gate per brain on average. This added complexity is a result of evolutionary dynamics: gates that improve $\mathcal{R}$ but not fitness are likely maintained under $\mathcal{R}$-augmentation but fall away due to genetic drift without. It may be that this added complexity contributes to the increase in task performance; once an additional gate is present and contributing to $\mathcal{R}$, it can also be more easily mutated to provide an increase in fitness as opposed to the unaugmented case where a gate would need to be created from scratch.

As a continuation of this investigation, I test if the improvement from $\mathcal{R}$-augmentation was caused by a change to the underlying network structure of the evolved Markov Brains. As mental representations and $\mathcal{R}$ require the persistence of information through looping constructs in the network, I examine the cyclical nature of the networks produced by unaugmented and augmented GAs by counting the number of self-referential information cycles of length 1 and length 2. There does not appear to be a substantive difference in the cyclical structures produced by either type of GA, see Figure 4.5. This measurement is only an ap-

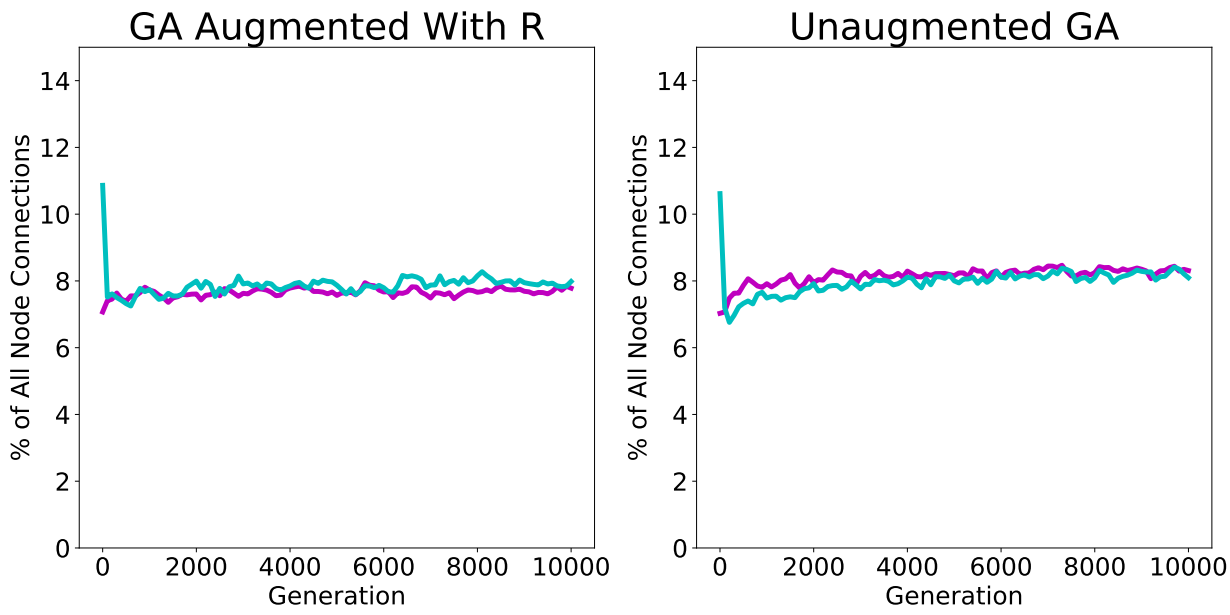

Figure 4.5: Proportion of 1- and 2- Cycles in Markov Brains produced by Augmented and Unaugmented GAs over evolutionary time. The purple lines represent the proportion of 1-cycles (e.g., Data is read from node A, processed by a gate, and then fed back into node A) among all node to node connections. The light blue lines represent the proportion of 2-cycles (e.g., Data is read from node B, processed by a gate, and fed into node C which is read, processed by a gate, and fed back into node B), against all node to node connections. Although not identical, the proportions both hover around the same value of 8 percent in networks produced by augmented and unagumented brains. ©2020 IEEE.

proximation and does not address how well the network uses any looping structures, which may explain why we detect no notable difference between networks created by augmented or unaugmented GAs.

After concluding that changes in network effects alone could not explain the change in performance from $\mathcal{R}$-augmentation, I examine how mutations resulted in changes to task fitness and $\mathcal{R}$ along the line of descent, hoping to find evidence of a change in the overall evolutionary dynamic.

Next, I examine the mutations by using the lines of descent from each replicate experiment from $\mathcal{R}$-augmented and unaugmented GAs. I measure the effect of each mutation on $\mathcal{R}$ and performance (W) independently using the difference between pairs of organisms on the line of descent. Thus, each mutation can be understood as a movement in a 2D space defined

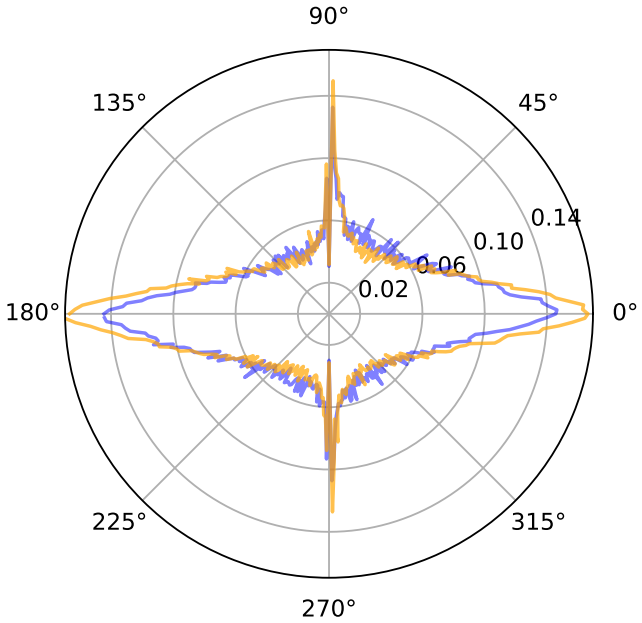

Figure 4.6: Mutational Changes in Markov Brains produced by Augmented and Unaugmented GAs. The blue line represents results from the lines of descent of GAs using $\mathcal{R}$-augmentation, while the orange line comes from unagumented GAs. Each point around the circle represents the average magnitude of the changes with a given proportion of change in $\mathcal{R}$ (vertical axis) and change in fitness (horizontal axis). The two distributions are significantly different (Kolmogorov–Smirnov 2-sample, $p < 0.0001$). ©2020 IEEE.

by two axes. I define the y-axis to be the change in R, and the x-axis to be the change in W. Neutral mutations do not cause any movements in this space, while an improvement in $\mathcal{R}$ corresponds to a move upwards and a decrease to a move downwards. Similarly, an improvement to W would result in a move to the right, and a decrease a move to the left. I first measure the average magnitude of the movements along each angle of these axes (see Figure 4.6). I also measure the number of mutations that moved along each angle of the axes, demonstrated in Figure 4.7.

The positive effect generated by $\mathcal{R}$-augmentation seems to be a result of change to the fitness landscape or more broadly a change in the evolutionary dynamics of the genetic algorithm. GAs augmented with $\mathcal{R}$ have more mutations that produce larger changes in both $\mathcal{R}$ and fitness, as opposed to unaugmented GAs whose mutations produce changes more exclusively in terms of task fitness. This difference was also seen in the count of the

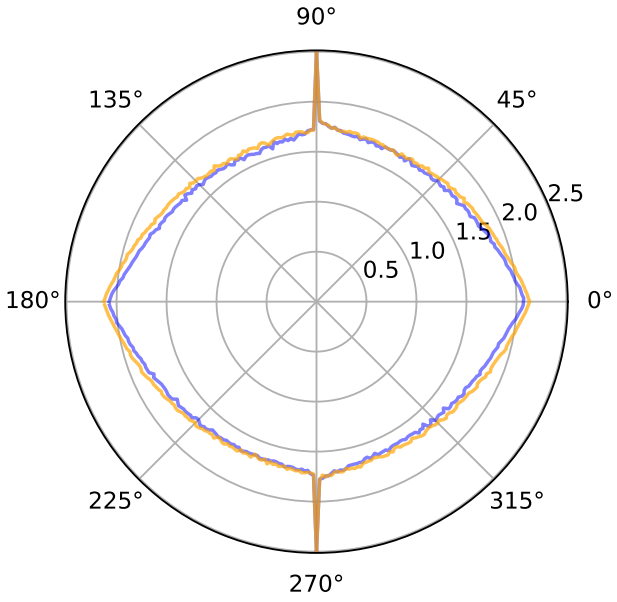

Figure 4.7: Mutational Changes in Markov Brains produced by Augmented and Unaugmented GAs. The blue line represents results from the lines of descent of GAs using $\mathcal{R}$-augmentation, while the orange line comes from unagumented GAs. Each point around the circle represents the average count of changes with a given proportion of change in $\mathcal{R}$ (vertical axis) and change in fitness (horizontal axis). The two distributions are significantly different (Kolmogorov–Smirnov 2-sample test, $p < 1 \times 10^{-7}$).

mutations, where fewer but perhaps more effective movements were made in the mutations $\mathcal{R}$-augmented GAs. These different mutations may allow the augmented GAs to explore the fitness landscape more efficiently, finding solutions that have better task performance in less evolutionary time.

4.5 Conclusion

I found that the improved performance found by $\mathcal{R}$-augmented GAs cannot be explained by changes to the structure of the evolved neural network, but could likely be explained by evolutionary dynamics effects. The mutations along the line of descent produce movement through the fitness landscape primarily in terms of task performance in unaugmented GAs, while augmented GAs have mutational effects that result in greater changes in terms of both $\mathcal{R}$ and task performance.

While there might be other possible factors at play causing the augmentation, a change to the fitness landscape appears likely, as the augmented fitness function is significantly altered by integrating R. Further, $\mathcal{R}$ measures a property that is desirable for the task–it is better for the agent to know more about the environment. As such, $\mathcal{R}$ and performance both have the chance to positively synergize in the augmented fitness function. That might not be true for other neuro-correlates or multiple parameter optimization. Thus I do not believe that this result necessarily generalizes to all other ways of combining a performance measure with a property of the optimized agent. However, further exploring the exact mechanisms of how $\mathcal{R}$-augmentation or other augmentation methods change the fitness function, and thus the evolutionary dynamics of the GA, could lead to more powerful GAs as a whole.

As there are other additional possible explanations contributing to the success of $\mathcal{R}$-augmentation, in the future I will explore other factors such as computational or behavioral changes to the optimized networks and their effect on $\mathcal{R}$-augmentation.

4.6 Acknowledgements

This work was in part sponsored by the BEACON Center for the Study of Evolution in Action NSF Cooperative Agreement No. DBI-0939454. I would like to thank Vincent Ragusa, Cliff Bohm, and Chris Adami for their insights and advice. All research was conducted using the MABE software platform, run on the Michigan State HPCC.

Chapter 5

$\mathcal{R}$-Augmentation maintains Performance as Sample Size Decreases

5.1 Abstract

Cognitive systems, both biological and artificial, require an understanding of the world in order to act and make decisions. This understanding is aided by the creation of mental models or representations. Although artificial cognitive systems created by deep learning struggle to form representations, previous work has shown that using neuro-evolution to generate representations in artificial cognitive systems is a viable method. This has been confirmed by using an information-theoretic measure, $\mathcal{R}$, to determine the amount of information a cognitive system has integrated about its environment. Furthermore, by modifying the fitness function to select for both $\mathcal{R}$ and task performance (through a process known as $\mathcal{R}$-augmentation), neuro-evolution can more quickly select for cognitive systems that have a high task performance. In this chapter, I extend this work to investigate a method of improving the efficiency of $\mathcal{R}$-augmentation by showing that using $\mathcal{R}$-augmentation requires only a small subset of the task cases to achieve high performance across the entire task. Furthermore, I

show that this improvement does not come at a cost to the ability of the cognitive system to generalize either within-domain or out-domain. This work has the potential to allow for more complex tasks and faster neuro-evolution of artificial cognitive systems.

5.2 Introduction

The field of Artificial Intelligence, and the sub-field of Machine Learning in particular, is continuously searching for better and more efficient ways to optimize neural networks to complete various tasks [Sun et al., 2019]. Many of these optimizations have focused on more efficient algorithms [Johnson and Zhang, 2013]; or on tweaking network architectures to more efficiently learn a specific problem (e.g., [Dao et al., 2020]). However, neither of these approaches address a fundamental problem with optimization for optimization's sake: namely, optimization alone does not necessarily produce a network with better understanding of any given task, and thus fails to produce networks that are able to generalize to similar tasks. For this chapter, I investigate how creating a better understanding of a specific task may allow for more efficient adaptation and better task performance.

In the psychological and philosophical literature [Phillips and Singer, 1997, Pinker, 2013], the ability of a cognitive system to understand a given environment or task is referred to as a mental model or mental representation. Improvements to the task performance of cognitive systems have been shown to correlate with improvements in the system's mental representations. More specifically, this mental representation is defined as a sensor-independent model of the environment stored within the system's memory. These representations may encapsulate an entire task or focus on individual concepts that the agent deals with [Halford, 2014]. More importantly, mental representations may allow an individual to assess, understand, and form internal models about the entirety of a task even though they are only exposed to a small subset of the entire possible experience. As such mental representations are desirable or even necessary for human intelligence [Collins and Gentner, 1987], recent work has attempted to

replicate these representations *in silico* [Edlund et al., 2011, Marstaller et al., 2013, Albantakis et al., 2014, Tehrani-Saleh et al., 2018, Olson et al., 2016, Tehrani-Saleh et al., 2019].

[Marstaller et al., 2013] developed an information-theoretic measure, referred to as $\mathcal{R}$, to measure these representations in a formal way . Formally, $\mathcal{R}$ is the information shared between the task or environmental states and the hidden states of the cognitive system, given the information from the sensor states. $\mathcal{R}$ correlates with task performance, indicating that it measures the cognitive system's understanding of the given task. Moreover, [Schossau et al., 2015] created a process of selecting for both the task performance and $\mathcal{R}$, referred to as $\mathcal{R}$-augmentation, to encourage the formation of representations in artificial cognitive systems. That paper also demonstrates that selecting for networks with better mental representations results in higher task performance in Markov Brains.

The core idea behind this chapter is simple; with a greater understanding of the task, as measured by $\mathcal{R}$ and promoted during evolution through $\mathcal{R}$-augmentation, I should be able to evolve cognitive systems towards high performance despite only viewing a few examples at any given time. This enhanced understanding should allow networks to understanding the missing pieces, and evolve the ability to complete the entire task. Furthermore, the ability of the cognitive system to generalize to tasks beyond what it has been evolved to solve should remain unchanged by the $\mathcal{R}$-augmentation process,

5.3 Methods

5.3.1 Mental Representations and $\mathcal{R}$

In order to quantify mental representations, I turn to information-theoretic analysis. More specifically, I focus on the information-theoretic measure $\mathcal{R}$ [Marstaller et al., 2013]. $\mathcal{R}$ is defined to be the information shared between the task and the cognitive system's memory that is not currently being provided by the sensors. This can be quantified in the theoretical

definition of Eq. 5.1, where T is the task variable, M is the random variable corresponding to the cognitive system's memory, and S is the variable corresponding to the states of the sensors.

$$\mathcal{R} = I(T : M|S) \tag{5.1}$$

Note that conditioning on sensor states is intended to isolate the information that has been integrated and stored in memory separate from current sensory experience, thus finding true representations. This conditioning prevents the identification of internal models that are purely reactive to sensory experience.

In practice, during the evaluation of the task, the relevant task states, the states of the input provided by the task to the cognitive system, and the memory states of the agent (after it has processed the given input) are all recorded for all time points of the evaluation. The memory states of the cognitive system are defined by the particular network structure, while the input states are also easily identified for any given task. However, the relevant task or environmental states must be carefully selected to identify salient concepts within the task. A task where an agent must find its way through a maze, for example, would likely encode the absolute direction to the goal location as well as a local optimum direction as salient concepts. For the salient features that I use in this work, see the task description in 5.3.2.

When all of the state vectors have been collected, the probability of each state is calculated. These probabilities are used in a series of Shannon entropy calculations to find the measurement of $\mathcal{R}$, as optimized previously (see Chapter 3). The total equation for $\mathcal{R}$ as used is shown in Eq. 5.2.

$$\mathcal{R} = H(S,T) + H(S,M) - H(S) - H(T,M,S)\ . \tag{5.2}$$

The joint entropies are calculated by appending the individual state vectors together over all time points and finding probabilities for the appended vectors. Then, similarly to the process for individual states, these probabilities are used to calculate joint entropies as required for Eq. 5.2. As the memory, task, and sensor states collected are often not a comprehensive enumeration of all possible states, $\mathcal{R}$ is often only an approximation of the true amount of representation that a cognitive system has. Previous work [Kirkpatrick and Hintze, 2019] shows that it is possible to measure $\mathcal{R}$ using only some of the task states. Given that $\mathcal{R}$ is an approximation, using fewer states to calculate $\mathcal{R}$ may make the approximation slightly less accurate, but this inaccuracy does not invalidate the measurement itself.

5.3.2 Active Categorical Perception Task

The Active Categorical Perception Task (ACP) is the primary task used in the study of mental representations and $\mathcal{R}$ in evolved artificial cognitive systems. The task is both complex enough that agents perform interesting behavior when evolved on the task and simple enough that the integral components of the task can be easily identified and quantified, and therefore the agent's understanding of those components can be readily measured. The task itself consists of an agent that sits underneath a series of falling blocks. The agent, controlled by the artificial cognitive system, has two sensors, each of width two, that are separated by a gap also of width two, and the agent can move to the right, to the left, or stay put on any given update (see Figure 5.1). The agent has two outputs, where if only the first output is set to 1 the agent moves left, if only the second output is set to 1 the agent moves right, and for all other combinations the agent remains stationary. The blocks that fall towards the agent move at a constant rate either to the left or to the right. The blocks may also be large or small, and the agent must navigate relative to the small blocks which must be caught and the large blocks which must be avoided. The agent is considered to be catching the block if any part of the agent overlaps with the block. Unless stated otherwise, the small blocks have a width of 2 units, and the large blocks have a width of 4 units. The block drops towards the

agent for 32 time steps, and exists in a toroidal world (i.e., where the edges wrap around) of width 16. With the agent starting at point zero by definition, this means that the block may drop from any of 16 different starting positions relative to the agent. By default the agent is tested on all possible permutations of block size, movement direction, and relative starting position, as enumerated in Figure 5.2.

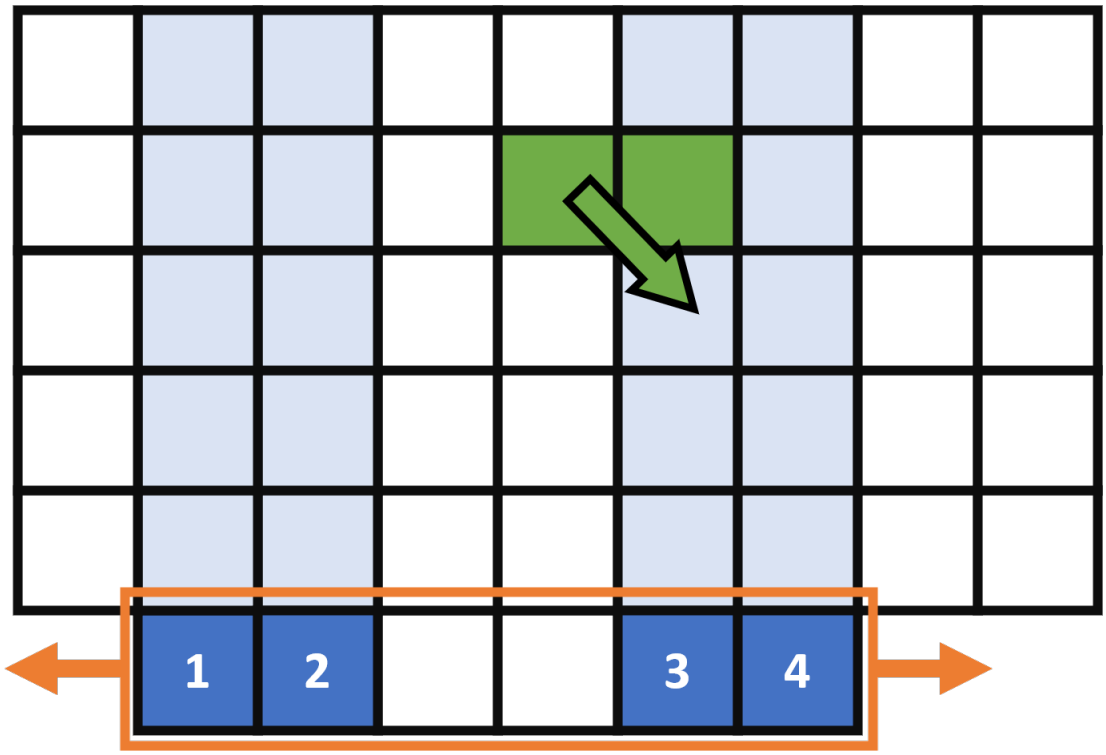

Figure 5.1: A subsection of a single case of the Active Categorical Perception Task. The agent, visualized by the orange box, can move either to the left or right as indicated by the orange arrows. The four sensors, labelled 1,2,3, and 4, can each see directly upwards as indicated by the light blue shading. A small block with width 2 is falling towards the agent and moving to the left, as indicated by the green arrow. At the current point in time, sensor 3 is the only sensor that would return a signal, while on the next update, assuming that the agent did not move, both sensors 3 and 4 would return a signal.

The agent is evaluated on the individual trials it sees by determining whether it correctly or incorrectly avoids or catches the block for that trial. Based on previous work [Marstaller et al., 2013], the fitness for this task is calculated by raising the difference of correct (C) and incorrect (I) actions over all trials to a given power (here 1.1). This equation for fitness W is formalized in Equation 5.3.

Block Size / Movement Direction	Block Starting Position															
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Small / Left	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Small / Right	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
Large / Left	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48
Large / Right	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64

Figure 5.2: All possible cases in the Active Categorical Perception Task. Each square in the rectangle indicates a specific combination of block size, movement direction, and starting position referred to as a case, labelled here from 1 to 64 to distinguish their uniqueness. For example the case labelled 20 is the case where the small block starts at position 3 (relative to the agent at position 0), and falls to the right.

$$W = 1.10^{(C-I)} \tag{5.3}$$

For the purposes of calculating $\mathcal{R}$, the relevant world states, as identified in previous literature (and reaffirmed in Chapter 2), are the size of the block (i.e., whether it is large or small), the position of the block relative to the agent (i.e., whether the block is over the agent or not), and whether the block is falling to the right or to the left. The sensor states used in calculating $\mathcal{R}$ are the values input to each of the four sensors at each time step.

5.3.3 Fitness Function Augmentation With $\mathcal{R}$

Recent work has identified a process of modifying the fitness function to encourage the evolution of representation. This process, referred to as $\mathcal{R}$-augmenation, is shown in [Schossau et al., 2015] to improve task performance relative to an unaugmented GA. The change from the original fitness function (Eq. 5.3) is best described as multiplying the original function by a term utilizing $\mathcal{R}$, which approximates a multi-objective function and encourages the simultaneous evolution of $\mathcal{R}$ and task performance. This may allow changes in $\mathcal{R}$, and hence improvements to the agent's understanding of its environment, to be selected for ahead of

when they are needed to improve performance, which would allow for the faster evolutionary adaptation (see Ch. 4). This selection for $\mathcal{R}$ before it is needed is a form of exaptation, where the original selection of $\mathcal{R}$ is intended to be reused later to increase task performance. The added term is a normalized value between 1 and 2, which allows for a multiplicative boost to the fitness. This value is achieved by adding 1 to a $\mathcal{R}$ value normalized to between 0 and 1, where the normalization occurs when $\mathcal{R}$ is divided by a task- and network-dependent theoretical maximum value of $\mathcal{R}$, denoted $\mathcal{R}_{max}$. This $\mathcal{R}$-augmented fitness function is shown in Eq. 5.4.

$$W = 1.10^{(C-I)}(1 + \frac{\mathcal{R}}{\mathcal{R}_{max}}) \tag{5.4}$$

5.3.4 Recurrent Neural Networks

Recurrent Neural Networks (RNNs) are a standard neural network type commonly used in a variety of applications. While they may take many forms, here I use a fully-connected, feed-forward, multi-layer network with genetically encoded weights. The recurrent nodes used are appended to the network's input and output layers, and persist between time steps. More specifically, there is a layer of input and recurrent nodes, one hidden layer of nodes that is reset between updates, and a layer of output and recurrent nodes. The activation function is the standard hyperbolic tangent function, applied to a summation of the weights times previous layer values. The recurrent nodes fulfill the role of internal memory for the calculation of $\mathcal{R}$.

5.3.5 Markov Brains

Markov Brains [Hintze et al., 2017] are sparse, evolvable neural networks commonly used for the study of mental representations in artificial cognitive systems among other applications. They consist of an input buffer that receives data from the environment, an output buffer that allows the network to provide data to the environment, a hidden buffer that store data,

and computational gates that read from and write to each of those sets of states. The hidden buffer recurs across time points, allowing for the preservation of memories. Sizes of the input and output buffers are determined by the task selected, and hidden buffers are determined by the user. For the ACP task, the input buffer is size 4, and the output buffer is size 2. Here, I select a hidden buffer size of 8 in line with previous literature. The gates, genetically encoded, read from a fixed set of 1-4 input or hidden nodes, and write to a set of 1-4 output or hidden nodes. While there exist many different types of gate [Hintze et al., 2019], here I only use two in the context of this chapter. The first is the deterministic logic gate, where a binary logic table is encoded that defines the output states provided for any given input. This is the gate type most commonly used in previous work. Also, I use the ANN gate which operates as a single hidden layer with functionality identical to the RNN, reading from a set of 1-4 genetically encoded input and hidden nodes as the prior layer, and writing to a set of 1-4 genetically encoded output and hidden nodes as the next layer. I construct Markov Brains only using one gate type or the other, with Markov Brains using deterministic gates referred to as simply Markov Brains, while Markov Brains using ANN gates are referred to as Markov Brain - ANN hybrids. The naming of the Markov Brain - ANN hybrids indicate their functionality, which has been shown in recent work [Albani et al., 2021] to approximate a sparse RNN. For the purposes of the calculation of $\mathcal{R}$, the hidden nodes are considered the internal memory.

5.3.6 Evolutionary Algorithm

All of the evolutionary runs and experiments used in this chapter follow the same pattern described here. I generate each network from a circular genome, with starting length of 5000 sites, maximum length of 20,000 sites, and minimum length of 2000 sites. There are three types of mutation allowed on these genomes, namely point, insertion duplication, and deletion. The point mutation changes a site changes to a different random value, with a per-site probability of 0.005. The insertion duplication mutation copies a section of the genome and inserts the

section at another point in the genome, which occurs with a per-site probability of 0.00002, copying a chunk of the genome between 128 and 512 sites long and inserting that chunk at a random point. The deletion mutation deletes a chunk of the genome between 128 and 512 sites long from the genome, with a per-site probability of 0.00002. Genomes and the resulting networks were generated randomly in initial populations of 100 organisms. I applied the fitness function for each individual only after evaluation on the ACP task and the subsequent calculation of $\mathcal{R}$, where the fitness function is either the unaugmented function (Eq. 5.3) or the $\mathcal{R}$-augmented function (Eq. 5.4). I then applied roulette (i.e., fitness-proportional) selection [Thomas, 1996] using the generated fitness values to choose a new population of size 100. I applied the three mutation operations (point, insertion duplication, and deletion) to the new population. I repeated this process for 10,000 generations for each experimental condition. At the end of evolution, I gather the agents along the line of descent [Lenski et al., 2003], and use the agents at the end of the line of descent for further analysis.

5.3.7 Reduction of Trials

The first major experiment of this chapter is the assessment of a standard GA compared to a GA using $\mathcal{R}$-augmentation across varying subset size of trials provided to the agent. For any given assessment of the ACP, there are 64 individual trials run, although for these experiments the agent is only presented with a random subset of those trials with size n, for selected $n \leq 64$. Using $n = 64$ would be equivalent to the standard evaluation of the ACP Task. As an example, for $n = 32$ on any given generation a random 32 trials are selected and used to evaluate the population (see Fig 5.3 for an example of such a subset). For each generation, I randomly re-select a different subset of the same size. This is similar in concept to lexicase selection [Helmuth et al., 2015] as a small random subset of the cases are used, although in contrast to lexicase selection here the number of cases used remains fixed and the selection method is a typical roulette selection. I run these experiments with each of the three brain types (Markov Brain, RNN, and Markov Brain - ANN Hybrid), a range of subset sizes ($n = 1$,

2, 4, 6, 8, 10, 12, 14, 16, 32, 48, 64), and using GAs with and without $\mathcal{R}$-augmentation. I evaluate the different network types to determine the applicability of the method to different cognitive structures. I selected the subset sizes to provide sufficient coverage around the portions of the graph where the results changed most drastically, without being computationally wasteful. For each experimental condition, I run 200 replicate experiments with other evolutionary parameters as described in 5.3.6. At the end of evolution, I evaluate the evolved networks on the entire test case set (i.e., all 64 cases). For the experimental conditions using Markov Brains, I also record the time of execution for the entire program when returned, although not all experimental runs return a notification. All experimental runs were conducted on the Michigan State University High Performance Computing cluster on the Intel-16 architecture.

Block Size / Movement Direction	Block Starting Position															
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Small / Left	E		E		E	E	E			E		E		E	E	
Small / Right	E	E				E				E	E		E		E	
Large / Left		E			E	E				E		E	E			E
Large / Right	E		E		E	E	E			E	E		E			E

Figure 5.3: A possible evaluative subset for the Reduction of Trials experiments with $n = 32$. Here, the 32 cases marked with an 'E' would be evaluated for correctness and used to generate the fitness and measure $\mathcal{R}$. Note that this is only one possible random selection, and any random selection is equally likely in practice.

5.3.8 Generalization

As a natural consequence of the primary experiment, where agents are expected to evolve to perform well on the entire task set despite only seeing a subset at any given time, I must also consider how this affects generalization. Generalization refers to the ability of a network to perform correctly on conditions that it has not seen before. There are two main forms of generalization: in-domain generalization and out-domain generalization. The difference

between these forms lies in the similarity of the new testing data to the data the network was trained or evolved under. When the new data is more similar to or comes from the same distribution as the training data, I consider it to be in-domain generalization. On the other hand, if the data is dissimilar to or comes from a different distribution than the training data, it is considered to be out-domain generalization.

5.3.8.1 In-Domain Generalization

For the purposes of this chapter, I consider in-domain generalization to be where the task parameters, in particular block size, are held constant. To enable the evaluation of generalization within the ACP task, I withhold a subset of the runs to serve as a testing set, and then use the remainder for fitness evaluation during evolution. As a extensive although perhaps not comprehensive assessment of the in-domain generalization, I examine four different withholding schemes that determine which blocks are part of the withheld subset. The first, and simplest scheme is the Random Scheme, where 16 cases are selected at random from the 64 permutations or cases of the ACP task (see Fig 5.4 for a visualization). In each experimental run I choose a different selection of cases for each run, but this set remains constant over evolutionary time. For the Starting Position Spaced scheme I withhold cases for 4 of the possible block starting position, with the distribution of those positions evenly spaced across the 16 possible positions (see Fig. 5.5). Next is the Starting Position Grouped Scheme, where again I withhold cases for 4 of the block starting positions, but the starting positions are grouped together in one chunk (see Fig. 5.6). Lastly, the One Block / Direction scheme is where I withhold all of the cases for one block size (e.g., small) moving in one direction (e.g., to the left; see Fig. 5.7).

For each of these four in-domain generalization schemes, I construct experiments where I use a standard GA or an $\mathcal{R}$-augmented GA, and also use varying subsets of the non-withheld sets for evolution in a manner identical to the reduction of trials experiments, although here I use $n = 3, 8, 13, 18, 23, 28, 33, 38, 43, 48$. For each experimental combination, 200 replicate

Block Size / Movement Direction	Block Starting Position															
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Small / Left		W					W	W							W	
Small / Right				W		W					W					W
Large / Left			W					W				W				
Large / Right		W	W			W				W			W			

Figure 5.4: Example Cases in Withheld subset under the Random Scheme. Each square in the rectangle indicates a specific combination of block size, movement direction, and starting position referred to as a case. The 16 cases marked with a 'W' are withheld for testing, while the remaining 48 unmarked cases are used for evaluation during evolution. The cases withheld under this scheme are 16 randomly selected cases.

Block Size / Movement Direction	Block Starting Position															
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Small / Left	W				W				W				W			
Small / Right	W				W				W				W			
Large / Left	W				W				W				W			
Large / Right	W				W				W				W			

Figure 5.5: Example Cases in Withheld subset under the Starting Position Spaced Scheme. Each square in the rectangle indicates a specific combination of block size, movement direction, and starting position referred to as a case. The 16 cases marked with a 'W' are withheld for testing, while the remaining 48 unmarked cases are used for evaluation during evolution. The cases withheld under this scheme are all of the cases for 4 different starting positions, where the starting positions are evenly spaced across the range of possible starting positions.

Block Size / Movement Direction	Block Starting Position															
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Small / Left	W	W	W	W												
Small / Right	W	W	W	W												
Large / Left	W	W	W	W												
Large / Right	W	W	W	W												

Figure 5.6: Example Cases in Withheld subset under the Starting Position Grouped Scheme. Each square in the rectangle indicates a specific combination of block size, movement direction, and starting position referred to as a case. The 16 cases marked with a 'W' are withheld for testing, while the remaining 48 unmarked cases are used for evaluation during evolution. The cases withheld under this scheme are all of the cases for 4 different starting positions, where the starting positions are grouped together.

Block Size / Movement Direction	Block Starting Position															
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Small / Left	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W
Small / Right																
Large / Left																
Large / Right																

Figure 5.7: Example Cases in Withheld subset under the One Block / Direction Scheme. Each square in the rectangle indicates a specific combination of block size, movement direction, and starting position referred to as a case. The 16 cases marked with a 'W' are withheld for testing, while the remaining 48 unmarked cases are used for evaluation during evolution. The cases withheld under this scheme are all of the cases for 1 block size and movement direction combination.

experiments are run with other evolutionary parameters as described in 5.3.6. At the end of evolution, the networks created are evaluated on the withheld test set (unique to each scheme).

5.3.8.2 Out-Domain Generalization

In order to study the capacity of networks for out-domain generalization, I turn to a straightforward modification of the task: manipulation of the block sizes that are present. More specifically, I would evolve the networks on one set of large and small blocks (e.g., width 2 and width 6), and then see if they can generalize to another set of blocks (e.g., width 3 and width 5). I focus on a subset of the possible combinations shown in Chapter 2, selected to investigate those combinations without a previously known bias towards $\mathcal{R}$-augmentation, so that I can isolate any effect on out-domain generalization caused by the reduction of trials. These combinations are where I evolve the networks on widths 2,6 and test their ability to generalize to widths 3,5, and where I evolve the networks on widths 3,5 and test their ability to generalize to widths 2,6, both using the standard agent setup as seen in [Marstaller et al., 2013]. For each of these two out-domain generalization setups, I construct experiments where I use a standard GA or an $\mathcal{R}$-augmented GA, and also use varying subsets of the test cases for evolution in a manner identical to the reduction of trials experiments. For each experimental condition, I run 200 replicate experiments with other evolutionary parameters as described in 5.3.6. At the end of evolution, I evaluate the evolved networks on the 64 cases of the other block pairing that was not used during evolution.

5.4 Results and Discussion

First, in order to evaluate the feasibility of the method for different network structures, I examine the results across decreasing sample size, for the 3 major brain types: Markov Brain with Deterministic Logic Gates (Figure 5.8), RNN (Figure 5.9, and the Markov Brain - ANN Hybrid (Figure 5.10). Each of these shows the average performance tested at end of

evolution across the entire test set, regardless of the subset size n used, evolved either with the $\mathcal{R}$-augmented GA, or with the unaugmented GA. I observe that for both the standard Markov Brain and the Markov Brain with ANN gates, $\mathcal{R}$-augmentation produces networks that perform at least as well as or better than those produced by an unaugmented GA, even when a smaller subset is used for the augmented GA. This result is seen most clearly in Markov Brains where networks created by the augmented GA for $n = 16$ achieve an average tested task performance across the whole set (92.4% correct) that is similar to what as the networks evolved using an unaugmented GA achieve at $n = 32$ (93.1% correct). That is, when using $\mathcal{R}$-augmentation, you can potentially attain the same level of performance while only using a quarter of the test cases to measure performance. A similar increase in performance at low values of n can be seen for the Markov Brain - ANN hybrid.

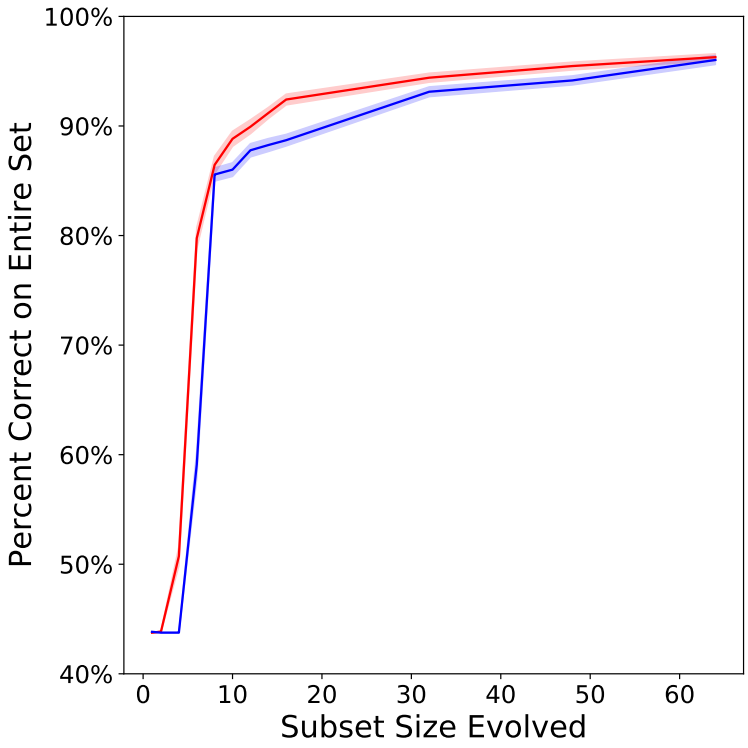

Figure 5.8: Performance at end of evolution under decreasing sample size for Markov Brain with Deterministic Logic Gates. The red line is the average performance of Markov Brains evolved using a GA augmented with $\mathcal{R}$, while the shaded area in red is the corresponding standard error. The blue line is the average performance of Markov Brains evolved using an unaugmented GA, and the shaded blue area is the corresponding standard error. The sample sizes used were $n = 1, 2, 4, 6, 8, 10, 12, 14, 16, 32, 48, 64$.

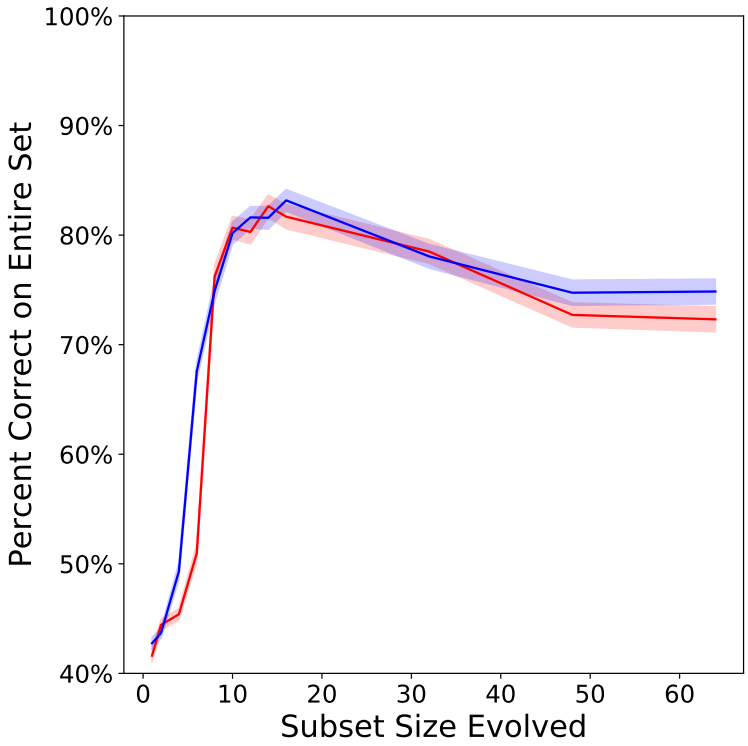

Figure 5.9: Performance at end of evolution under decreasing sample size for RNN. The red line is the average performance of RNNs evolved using a GA augmented with $\mathcal{R}$, while the shaded area in red is the corresponding standard error. The blue line is the average performance of RNNs evolved using an unaugmented GA, and the shaded blue area is the corresponding standard error. The sample sizes used were $n = 1, 2, 4, 6, 8, 10, 12, 14, 16, 32, 48, 64$.

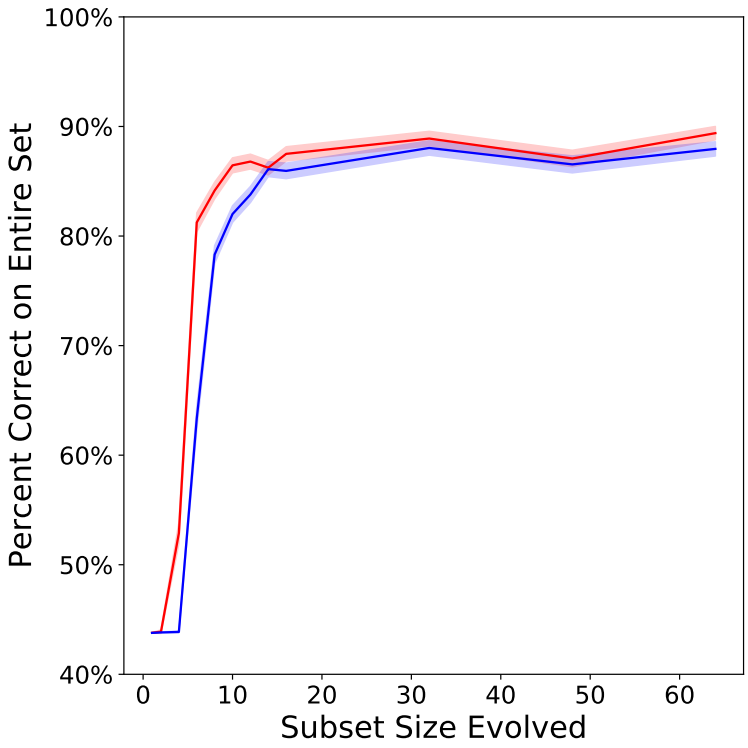

Figure 5.10: Performance at end of evolution under decreasing sample size for Markov Brain with ANN Gates. The red line is the average performance of Markov Brain - ANN hybrids evolved using a GA augmented with $\mathcal{R}$, while the shaded area in red is the corresponding standard error. The blue line is the average performance of Markov Brain - ANN hybrids evolved using an unaugmented GA, and the shaded blue area is the corresponding standard error. The sample sizes used were $n =$ 1, 2, 4, 6, 8, 10, 12, 14, 16, 32, 48, 64.

The RNN does not see the same performance boost from the $\mathcal{R}$-augmentation process, regardless of sample size, but instead performs at approximately the same level when optimized by either the augmented or unaugmented GAs. A notable difference is that both GAs fail to optimize RNNs to achieve comparable task performance to the other structures either with or without $\mathcal{R}$-augmentation. Both the Markov Brain and Markov Brain - ANN hybrid achieve close to 90% correct or better, whereas the RNN reaches a maximum of less than 85% at its peak. This difficulty in achieving peak performance as opposed to other cognitive substrates may explain the different response to $\mathcal{R}$-augmentation. Most of the difference in peak performance can likely be explained by the relative structure of the RNN versus the Markov Brain and Markov Brain - ANN hybrid. RNNs are fully connected, as opposed to the sparse network connections of the Markov Brain and the Markov Brain - ANN hybrid. Previous work (e.g., [Hintze et al., 2018, Kirkpatrick and Hintze, 2019] and Ch. 3) has demonstrated that fully connected networks evolve differently from sparse networks. More specifically, fully-connected networks have preexisting information flow due to their connectivity and must reduce the connections to form useful representations. On the other hand, sparse networks need to evolve connections to create information flow and form representations. In this light, the benefits of $\mathcal{R}$-augmentation should and do have a greater effect on the sparse network types.

Another point of interest is that the RNN achieves peak performance over the entire set when it is only evaluated on between an eighth and a quarter of the whole set. Given that there are more connections and weights to evolve in the RNN as opposed to its sparsely-connected relatives, it seems likely that the full-connectivity struggles to produce results on all of the cases at a single time. However, when evaluated on only a subset of the cases, the network is able to produce more meaningful results, and evolutionary pressures are more effective in directing networks towards high performance.

I will now test how these additional analyses affect the computational time required to evolve with or without $\mathcal{R}$-augmentation. For any given evolutionary experiment, using $\mathcal{R}$-

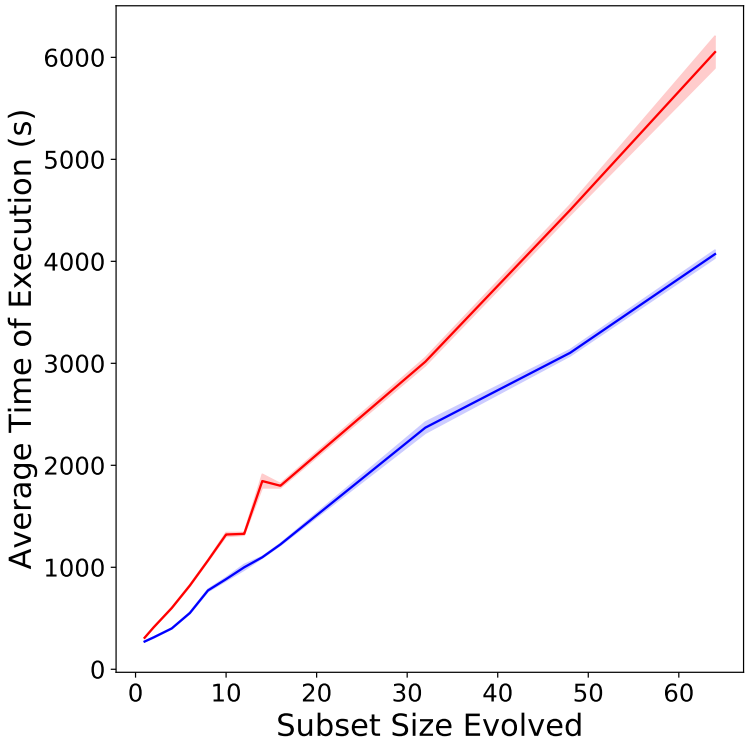

Figure 5.11: Average Execution Time Reported under decreasing sample size for Markov Brain with Deterministic Logic Gates, with and without $\mathcal{R}$-augmentation. The red line is the average execution time of runs using a GA augmented with $\mathcal{R}$, while the shaded area in red is the corresponding standard error of those times. The blue line is the average execution time of runs using an unaugmented GA, and the shaded blue area is the corresponding standard error of those times. The sample sizes used were $n = 1, 2, 4, 6, 8, 10, 12, 14, 16, 32, 48, 64$.

		GA Type	
		$\mathcal{R}$-Augmented	Unaugmented
Subset Size	1	148	155
	2	143	146
	4	125	146
	6	113	126
	8	137	128
	10	140	132
	12	140	133
	14	111	150
	16	115	127
	32	146	155
	48	152	142
	64	122	135

Table 5.1: Number of cases reporting execution times for each condition. Conditions were determined by subset size (n = 1, 2, 4, 6, 8, 10, 12, 14, 16, 32, 48, 64) and GA type (with and without $\mathcal{R}$-augmentation). Each condition had 200 replicate experiments run, however notifications were not received for each replicate.

augmentation incurs some cost due to the calculation of $\mathcal{R}$. I validate this by collecting the total execution time from the experimental replicate runs for the Markov Brains. Notifications were returned containing execution time data for approximately two-thirds of the experiments (3,267 of 4,800 experimental runs), distributed across the 24 conditions. The exact number of data points recorded for each condition is shown in Table 5.1; I average at least 110 times for each condition. The averages of these times for each condition is shown in Figure 5.11. Note that for all subset sizes, the $\mathcal{R}$-augmented GA requires more computational time than the unaugmented GA. However, given the reduction of trials shown in Figures 5.8 and 5.10, the networks created by a $\mathcal{R}$-augmented GA can still outperform those created by an unaugmented GA in a lesser amount of computational time. For example, the augmented GA ran for an average of approximately 1800 seconds with $n = 16$, while the unaugmented GA ran for an average of approximately 2370 seconds for $n = 32$, while achieving comparable performance. As such, the extra cost of calculating $\mathcal{R}$ may be outweighed by the reduction in trials; that is, an $\mathcal{R}$-augmented GA may be able to evolve higher-performing networks in less computational time than an unaugmented GA because the $\mathcal{R}$-augmented GA can use

fewer cases during evolution.

The remainder of the results presented here focus on validating the performance of Markov Brains evolved under varying subset sizes, primarily focusing on the ability of the agent to generalize to different data as previously studied in Chapter 2. I expand upon that work to include a more thorough investigation of in-domain vs. out-domain generalization, whereas previous work (Ch. 2) focused only on out-domain generalization. First, I focus on the four versions of in-domain generalization: withholding a random subset of the cases, withholding all of one block/direction combination, withholding a block of starting positions, and withholding a distributed sample of starting positions.

The generalization results of first pattern of in-domain withholding, random subset withholding, are seen in Figure 5.12. The data shown is the average performance on the withheld subset at end of evolution; in this case, the subset is randomly selected in each trial, with the caveat that the withheld subset is kept constant over evolutionary time. The results seen here track with the pattern seen in the first three reduction of trials results. That is, networks generated by an $\mathcal{R}$-augmented GA typically perform better on average during the final analysis (in this case, on the withheld subset). The performance difference is again particularly noticeable when $n < 20$.

The second of the in-domain withholding methods, namely withholding all of one block, is shown in Figure 5.13. Unlike the previous in-domain scheme, this withholding pattern prevents the cognitive system from learning anything about the block-direction pair, which seems to serve as a major impediment to generalizing to that withheld set. Performance for networks produced by both $\mathcal{R}$-augmented and unaugmented GAs suffer the same impediment, and their ability to generalize in this case seems near-identical.

Figure 5.14 shows the performance of Markov Brains tested on a withheld subset where the withheld cases have the same starting positions, and the starting postions are evenly distributed. In other words, given that I withhold 4 starting positions, each of the withheld

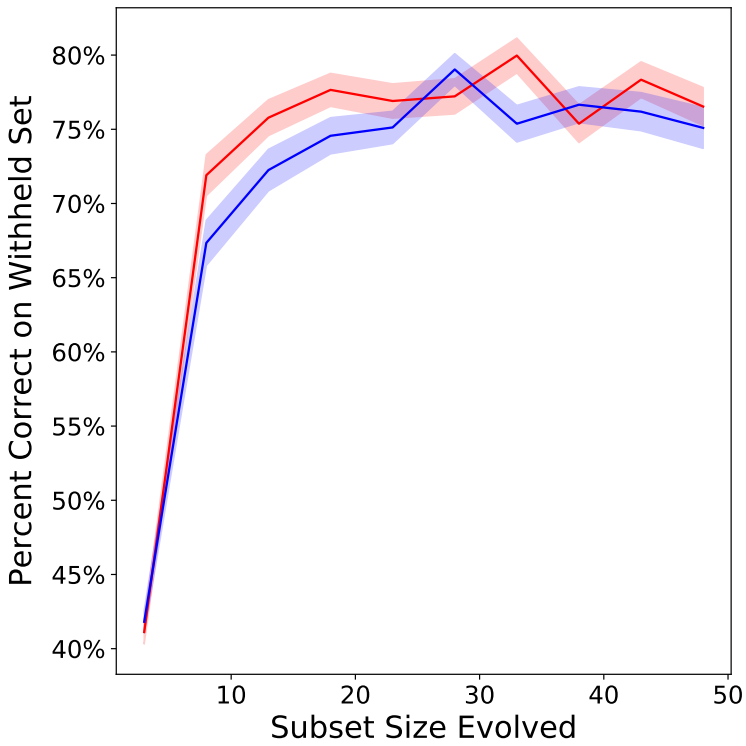

Figure 5.12: Performance tested on withheld group at end of evolution under decreasing sample size with random subset removed. The red line is the average performance of Markov Brains evolved using a GA augmented with $\mathcal{R}$, while the shaded area in red is the corresponding standard error. The blue line is the average performance of Markov Brains evolved using an unaugmented GA, and the shaded blue area is the corresponding standard error. The sample sizes tested were $n = 3$, 8, 13, 18, 23, 28, 33, 38, 43, 48.

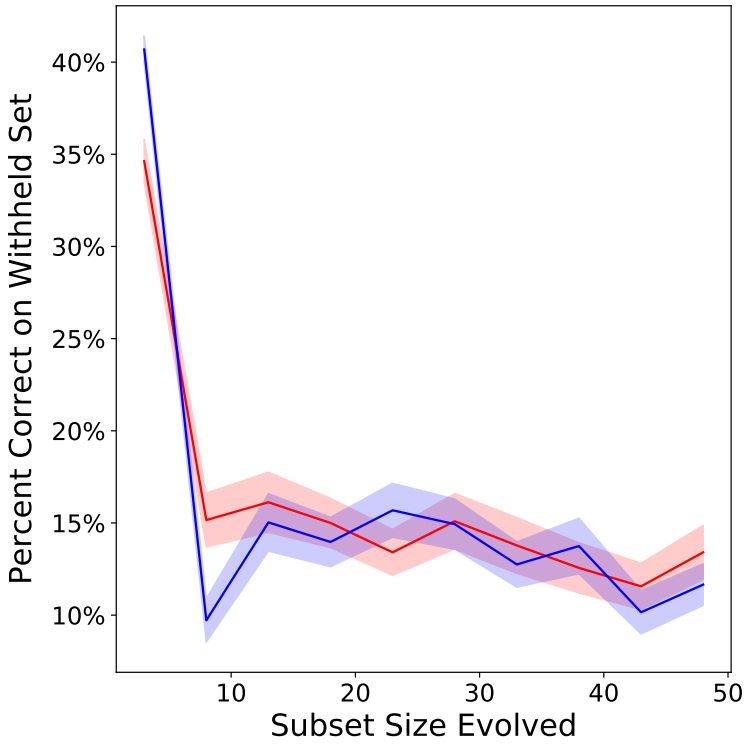

Figure 5.13: Performance tested on withheld group at end of evolution under decreasing sample size with all of 1 block type and direction withheld. The red line is the average performance of Markov Brains evolved using a GA augmented with $\mathcal{R}$, while the shaded area in red is the corresponding standard error. The blue line is the average performance of Markov Brains evolved using an unaugmented GA, and the shaded blue area is the corresponding standard error. The sample sizes tested were $n = 3, 8, 13, 18, 23, 28, 33, 38, 43, 48$.

positions is 4 units away from the other. For this withholding pattern, I see no notable difference between the ability to generalize of the Markov Brains evolved using an $\mathcal{R}$-augmented and those evolved using a standard GA. This pattern is notable because the agents have the highest generalization performance as compared to the other withholding patterns.

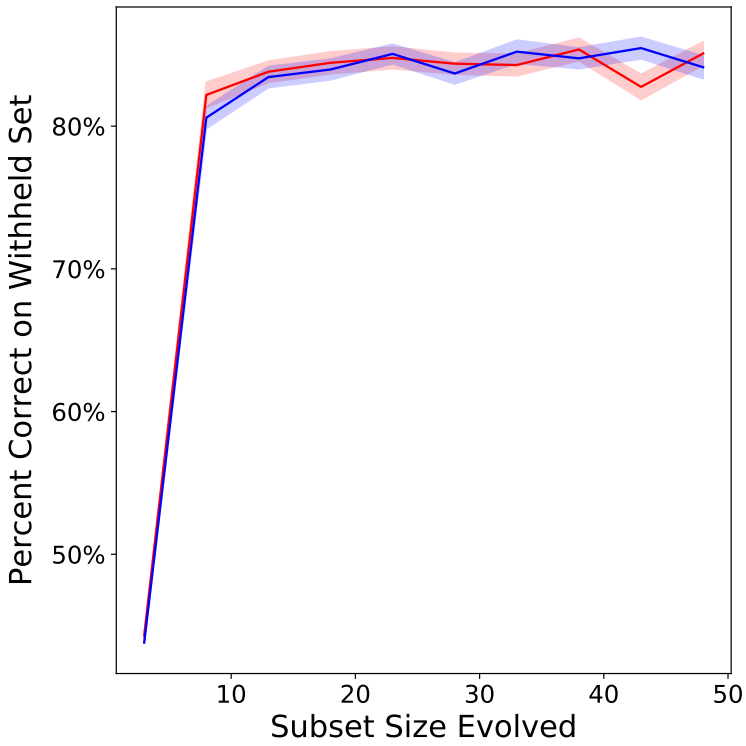

Figure 5.14: Performance tested on withheld group at end of evolution under decreasing sample size with all of 4 different start locations (evenly spaced) withheld. The red line is the average performance of Markov Brains evolved using a GA augmented with $\mathcal{R}$, while the shaded area in red is the corresponding standard error. The blue line is the average performance of Markov Brains evolved using an unaugmented GA, and the shaded blue area is the corresponding standard error. The sample sizes tested were $n = 3, 8, 13, 18, 23, 28, 33, 38, 43, 48$.

The last withholding pattern that I consider is where all of a group of starting positions are withheld, with the difference that the starting positions are from a contiguous block. The performances measured on the withheld set, as shown in Figure 5.15, suggest that Markov Brains evolved using $\mathcal{R}$-augmentation produce a higher average performance than those evolved with a unaugmented GA. This is particularly noticeable for $n < 18$, where the difference is often 5% or more.

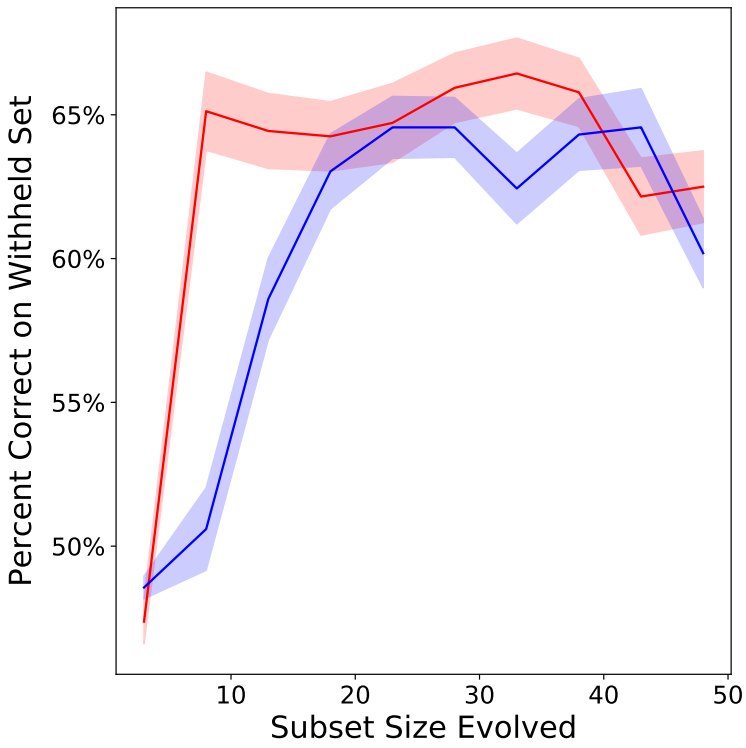

Figure 5.15: Performance tested on withheld group at end of evolution under decreasing sample size with all of 4 different start locations (grouped together) removed. The red line is the average performance of Markov Brains evolved using a GA augmented with $\mathcal{R}$, while the shaded area in red is the corresponding standard error. The blue line is the average performance of Markov Brains evolved using an unaugmented GA, and the shaded blue area is the corresponding standard error. The sample sizes tested were $n = 3, 8, 13, 18, 23, 28, 33, 38, 43, 48$.

In summary, the in-domain generalization capability of networks produced either with or without $\mathcal{R}$-augmentation seem largely identical. In the few cases where there is a notable difference, particularly at lower sample size, the mean performance of the networks produced by $\mathcal{R}$-augmented GAs is higher than that of networks produced by unaugmented GAs (see Fig. 5.12 and 5.15). These results are consistent with the results in Chapter 2, where $\mathcal{R}$-augmented GAs produce networks that generalize at least as well as or better than networks produced by unaugmented GAs. Interestingly, the one withholding pattern which prevents agents from seeing all of a block/direction pair and is therefore closest to out-domain generalization leads to very low performance measured on the withheld set. In either case, the ability of Markov Brains to generalize in-domain is not negatively affected by the $\mathcal{R}$-augmentation process.

Finally, I look at the ability of networks evolved with different subset sizes to generalize out-domain. I evolved Markov Brains using GAs with and without $\mathcal{R}$-augmentation, for a small block size of 2 and a large block size of 6, and also a small block size of 3 and a large block size of 5. At the end of evolution, the resulting cognitive systems were tested on the opposite block pairing (i.e., the networks evolved with the 2,6 pairing were tested on 3,5 and vice-versa). The results of these tests on the opposite sizing are visualized in Figures 5.16 and 5.17 for the 2,6 to 3,5 and 3,5 to 2,6 combinations, respectively. The mean difference between the performance of Markov Brains evolved using unaugmented and $\mathcal{R}$-augmented GAs is less than 3% in almost all cases. As such, I conclude that the out-domain ability of Markov Brains to generalize to unseen data is not negatively affected by $\mathcal{R}$-augmentation.

5.5 Conclusion

The results presented in this chapter offer a tantalizing opportunity for the application of $\mathcal{R}$-augmentation. The original promise of $\mathcal{R}$-augmentation relied on the fact that even though there was a computational cost to calculating $\mathcal{R}$, the improved performance benefits could still outweigh that extra computational cost. With this work, I demonstrate that it may be

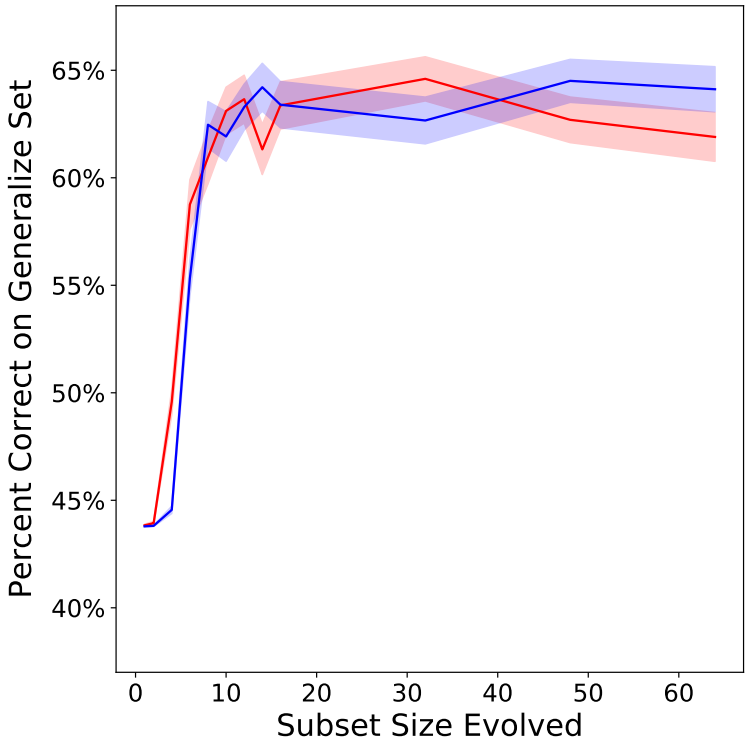

Figure 5.16: Performance of Markov Brains tested on blocks 3 and 5 at end of evolution under decreasing sample size when evolved on blocks 2 and 6. The red line is the average performance of Markov Brains evolved using a GA augmented with $\mathcal{R}$, while the shaded area in red is the corresponding standard error. The blue line is the average performance of Markov Brains evolved using an unaugmented GA, and the shaded blue area is the corresponding standard error. The sample sizes used were $n = 1, 2, 4, 6, 8, 10, 12, 14, 16, 32, 48, 64$.

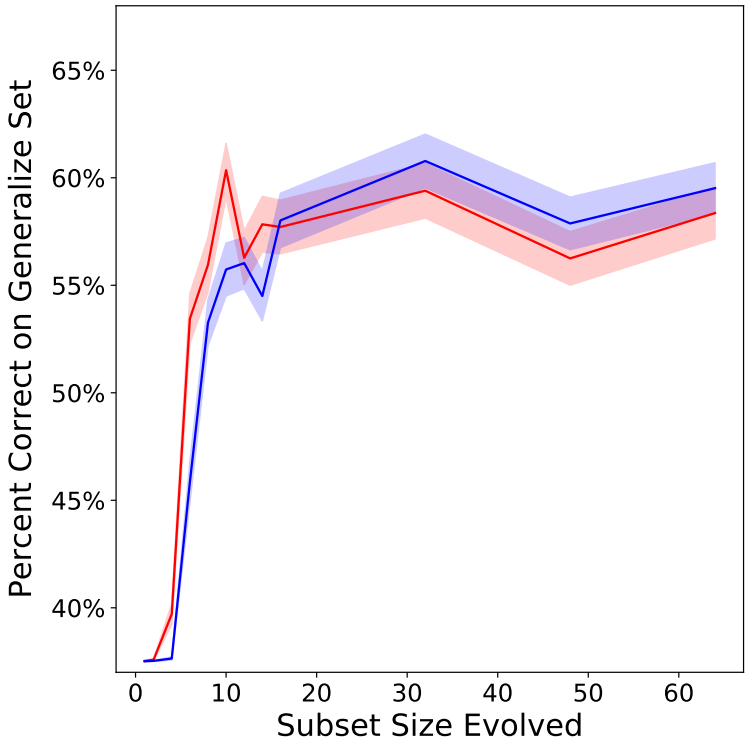

Figure 5.17: Performance tested on blocks 2 and 6 at end of evolution under decreasing sample size when evolved on blocks 3 and 5. The red line is the average performance of Markov Brains evolved using a GA augmented with $\mathcal{R}$, while the shaded area in red is the corresponding standard error. The blue line is the average performance of Markov Brains evolved using an unaugmented GA, and the shaded blue area is the corresponding standard error. The sample sizes used were $n =$ 1, 2, 4, 6, 8, 10, 12, 14, 16, 32, 48, 64.

possible to have your cake and eat it, too: you can evolve with $\mathcal{R}$-augmentation, and achieve comparable performance while minimizing execution time by reducing the number of trials during evolution.

Furthermore, this method of reducing computational time does not significantly impede the cognitive system's ability to generalize to other problems, either in-domain or out-domain. In fact, in some cases, the increased value of $\mathcal{R}$ (and understood increase in mental representation) caused by $\mathcal{R}$-augmentation may allow for a better understanding of the problem, resulting in increased ability to generalize.

Although this research was conducted primarily on a simple task, using evolutionary adaptation, I see potential applications that reach far beyond this original remit. For more complex tasks, particularly those with simulated physics governing object behavior or where 3-dimensional vision is used, the computational run-time of the task itself can rise exponentially. For such a task, where assessment can take hours or days, a case reduction of one half would allow for longer or more replicate experiments, assuming that the relevant environmental states can be encoded efficiently.

This work also hints at the applicability of $\mathcal{R}$-augmentation to other optimization methods, particularly machine learning. Given that for some tasks (e.g., image segmentation), the network only sees a portion or subset of the whole image and is expected to understand broader context, I see an immediate parallel. $\mathcal{R}$-augmentation applied to machine or deep learning models would take the form of a modification to the loss function instead of the fitness function, and could reduce the number of epochs needed to achieve high task performance.

Regardless of the direction in which the methods developed here are applied, there remains a number of open questions to be solved, ranging from how best to apply these methods to machine learning to understanding why the $\mathcal{R}$-augmentation process does not work as effectively in fully connected RNNs. While these results are promising, further refinement will be needed to allow for wider application and use. The results and methods here provide

a new starting point for the efficient creation of more capable cognitive systems.

Chapter 6

Using MAP-Elites to direct the evolution of desired neural characteristics

This chapter was originally published as "Douglas Kirkpatrick, & Arend Hintze (2021, July). Using MAP-Elites to direct the evolution of desired neural characteristics. In Proceedings of the ALIFE 2021: The 2021 Conference on Artificial Life. ALIFE 2021: The 2021 Conference on Artificial Life (pp. 113). ASME." Used with permission. Several additions and edits have been made, as well as changes to fit the tone and style of this dissertation.

6.1 Abstract

Artificial cognitive systems (e.g., artificial neural networks) have taken an ever more present role in the modern world, providing enhancements to everyday life in our cars, in our phones, and on the internet. In order to produce systems more capable of achieving their designated tasks, previous work has sought to direct the evolution of networks using a process referred to as $\mathcal{R}$-augmentation. This process selects for the maximisation of an information-theoretic

measure of the agent's stored understanding of the environment, or its representation ($\mathcal{R}$) in addition to the task performance. This method was shown to induce increased task performance in a shorter amount of evolutionary time compared to a standard genetic algorithm. Extensions of this work have looked at how $\mathcal{R}$-augmentation affects the distribution of representations across the neurons of the brain "tissue" or nodes of the network, referred to as smearedness ($\mathcal{S}$). Here I seek to improve upon the prior methods by moving beyond the simple maximization used in the original augmentation formula by using the MAP-Elites algorithm to identify intermediate target values to optimize towards. I also examine the feasibility of using MAP-Elites itself as an optimization method as opposed to the traditional selection methods used with $\mathcal{R}$-augmentation, to mixed success. These methods will allow us to shape how the network evolves, and produce better-performing artificial cognitive systems.

6.2 Introduction

While recent interest, funding, and computational breakthroughs have driven artificial intelligence to new heights in both capability and widespread use, there is still a lack of understanding on how to best shape the behavior, structure, and performance of artificial cognitive systems to reflect modern needs. In order to broadly adapt artificial cognitive systems to a wide range of tasks and problems, we should focus our efforts on generating cognitive systems that understand the tasks that they are presented with. For inspiration as to how to generate artificial cognitive systems that possess such understanding, I seek inspiration from beyond computer science. In the fields of philosophy, psychology, and cognitive science, for example, researchers have stressed the importance of mental representations. In these fields, representations are the internal imagery of objects that are not currently perceived by the sensory organs. According to cognitive scientists (and many philosophers, e.g., [Phillips and Singer, 1997, Pinker, 2013]) these internal models of the world are essential for intelligent decision-making, and form the very foundation of human intelligence [Collins and Gentner,

1987]. While the importance of mental representations for Artificial Intelligence (AI) has been clear from the outset, it has been proven difficult to program mental representations into computational substrates, leading proponents at the forefront of AI research to search for "Intelligence without Representation" [Brooks, 1991], an approach that has largely failed to generate truly intelligent systems. Instead, modern AI research has focused Convolutional Neural Networks (CNNs), which appear to be powerful classifiers of complex contextual visual scenes [LeCun et al., 1998]. However, many of the researchers that were spearheading the "Deep Learning" revolution now concede that those networks are vulnerable to so-called "adversarial perturbations" [Jo and Bengio, 2018], and are unlikely to represent a step towards artificial general intelligence, in part because they do not have robust mental models.

Recent research has shown that instead of directly programming mental representations, it is possible to produce them by evolving artificial brains that control agents that act and behave in complex simulated environments [Edlund et al., 2011, Marstaller et al., 2013, Albantakis et al., 2014, Tehrani-Saleh et al., 2018, Olson et al., 2016, Tehrani-Saleh et al., 2019]. Within these digital brains (Markov Brains, see [Hintze et al., 2017]) mental representations are used to make decisions in conjunction with (and sometimes even without) reference to sensory information. The ability to mathematically define representations in terms of information theory [Marstaller et al., 2013] has made it possible to monitor the evolution of representations quantitatively, and to measure what is being represented in the brain even where those representations are to be found in subsets of neurons. As it turns out, the relative structure of representations (i.e., how localized or distributed representations are) can be quantified as well. [Hintze et al., 2018] introduced a measure of smearedness ($\mathcal{S}$) that quantifies how much representations are localized within a single (or small group) of neurons, or conversely how much representations are smeared out over large groups of neurons. In previous work I showed that how smeared-out representations are depends on the type of cognitive system (see Chapter 3) and additionally this smearedness can affect the robustness of the computation [Kirkpatrick and Hintze, 2019].

Furthermore, previous work shows that rewarding representations (referred to as $\mathcal{R}$) explicitly (along with other aspects of fitness) in a Genetic Algorithm (GA) leads to significantly better performance because "augmenting evolution with $\mathcal{R}$" encourages the evolution of AIs with robust mental models (see [Schossau et al., 2015] or Chapter 2). However, many open questions remain. Is there an optimal balance between rewarding "R" and rewarding fitness? Should representations be modular (different concepts represented in different areas of the brain) or should they be non-local, "smeared out" over the brain, as it were? Here I focus on the role and impact of smearedness and how it can be used to augment genetic search.

One might assume that, like $\mathcal{R}$-augmentation, increasing smearedness is also advantageous. I will show that this is not the case, and that trying to minimize it is equally useless. Viewed in the context of neuro-correlate maximization introduced by [Schossau et al., 2015], the smearedness $\mathcal{S}$ will not help us to improve the performance of the GA. However, I will show that integrating the neuro-correlates $\mathcal{S}$ and $\mathcal{R}$ with MAP-Elites optimization [Mouret and Clune, 2015] produces a novel, useful method to investigate this issue.

MAP-Elites was originally designed to improve performance by maintaining a diverse population of solutions. MAP-Elites optimization does not allow all solutions in the population to compete against each other, but instead creates a grid of solutions. The grid is superimposed on a multidimensional parameter space, where each parameter describes set of solutions binned by their performance on that parameter. The best performance for different combinations of parameter values are stored, creating a "Map of Elites". For example, instead of just evolving a predator for hunting success, one could also measure predator size and weight. When evaluating a predator's performance, the size and weight of the predator are also determined and are used to determine a bin on the grid that it is categorized into. This predator is then not competing against the rest of the population, but only against a previous solution with similar size and weight defined by the grid. When applying the MAP-Elites method using $\mathcal{R}$ and $\mathcal{S}$ as the dimensions for the grid, the search of the GA can be improved. Here I find that, perhaps as expected, the optimal solutions have an intermediary $\mathcal{S}$ rather than a maximal

or minimal $\mathcal{S}$. However, surprisingly, those best solutions do not maximize $\mathcal{R}$ but are found to have intermediary values of $\mathcal{R}$ as well. This shows, that while maximizing or minimizing $\mathcal{S}$ does not improve the search of a GA, $\mathcal{S}$ is still a useful neurocorrelate when applying MAP-Elites selection. Further, using $\mathcal{S}$ in conjunction with $\mathcal{R}$ in MAP-Elites outperforms $\mathcal{R}$-augmentation in some cases.

6.3 Methods

6.3.1 Active Categorical Perception Task

The Active Categorical Perception Task (ACP) is the preeminent task used for evolution of artificial agents (especially Markov Brains) for the assessment of mental representations and $\mathcal{R}$ [Marstaller et al., 2013, Schossau et al., 2015, Hintze et al., 2018]. In this task, an agent equipped with a sensor made from four sensor neurons arranged in two groups of two (with a blind spot in between) must either catch or avoid blocks falling diagonally at constant rate, depending on the size of the block (see Figure 6.1). Blocks of varying width (here, width 2 or width 4) are dropped one at a time towards the agent on a toroidal surface of height 34 units and width 16 units, with the blocks moving by 1 unit to the left or to the right as they fall 1 unit down towards the agent. The agent must identify the size and direction of the falling block, and position itself to either be under (i.e., catch) the small blocks or not under (i.e., avoid) the large blocks. As the agent cannot see the whole block at one point in time due to the blind spot, it must integrate information from successive timesteps in order to decide which actions to take. Given the sizes of the blocks, the movement direction, and 16 different starting positions relative to the agent, there are 64 different conditions that the agent is comprehensively evaluated for. Agents are assigned fitness based on the number of correct (C) and incorrect (I) catch/avoid decisions that they make, using an exponential fitness function (see Equation 6.1).

$$W = 1.10^{(C-I)} \tag{6.1}$$

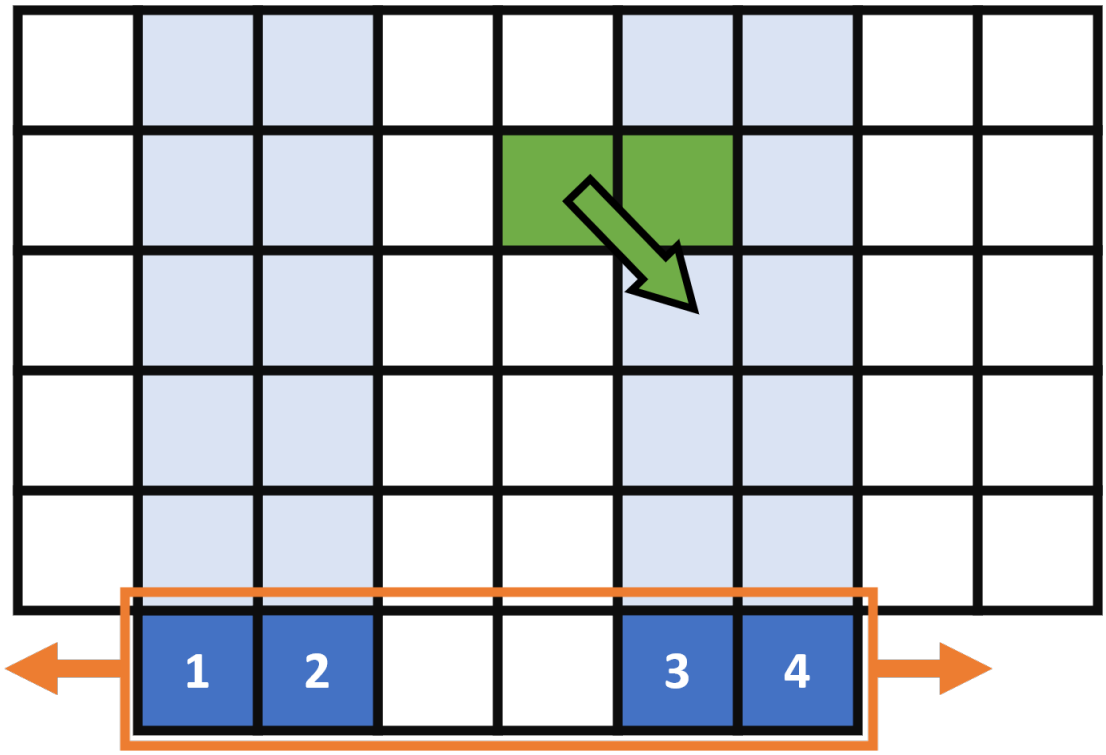

Figure 6.1: A subsection of a single case of the Active Categorical Perception Task. The agent, visualized by the orange box, can move either to the left or right as indicated by the orange arrows. The four sensors, labelled 1,2,3, and 4, can each see directly upwards as indicated by the light blue shading. A small block with width 2 is falling towards the agent and moving to the left, as indicated by the green arrow. At the current point in time, sensor 3 is the only sensor that would return a signal, while on the next update, assuming that the agent did not move, both sensors 3 and 4 would return a signal.

6.3.2 Numerical Comparison Task

The Numerical Comparison Task (NCT) is derived from real-world psychological [Merritt and Brannon, 2013] and biological [Nieder, 2018, Merritt et al., 2009] experiments with demonstrated utility in measuring $\mathcal{R}$ experimentally *in silico* [Kirkpatrick and Hintze, 2019]. In short, the agent receives one number and waits for a number of updates over which the agent is expected to remember the first number. After that time period the agent is given a second number, and again waits for the same number of updates, after which the agent must report

which of the numbers was larger. *In silico*, the agent has a number of inputs, here 5, which are set to be either 0 or 1. The agent must keep track of how many of these inputs are provided as 1s (i.e., remember the number of 1s). There are a number of intermediary brain updates (here 3) after which the inputs are reset to have a different number of 1s. After a second intermediary period of brain updates (where length equal to the first period), the agent must provide an output with a value of 1 if the second input was larger than the first value, and a 0 otherwise. During the intermediary period, the agent may be provided with a constant signal (all 1s or 0s) or the input bits may be assigned randomly. For the experiments described herein, the agent was provided with random noise during the intermediary timesteps, with a bit being randomly assigned with probability $p = 0.5$. The agents are tested over all possible permutations of input variations (i.e., the value of 2 could be input as 01100 or 11000, etc.) as well as permutations of relative number comparisons where unequal values are compared (i.e., 1 vs. 2, 2 vs. 1, 2 vs 5, etc.). I show an example of an agent being tested on a single permutation in Figure 6.2. The NCT is structured so that agents must receive information at two points in time, and then make a decision about them at a third time point, requiring information integration to occur between all three time points. Similarly to the ACP, agents are given fitness exponentially relative to the number of correct (C) and incorrect (I) decisions they make regarding which number is larger, again using the fitness function from Equation 6.1.

6.3.3 Markov Brains

Markov Brains are a sparsely-connected evolvable neural network structure. These networks interact with their environment through sensory and motor neurons (also called nodes), which can receive input from and provide output to their environment, respectively. The number of input and output neurons in each Markov Brain is task-dependent. Here I use 4 input neurons and 2 output neurons for the Active Categorical Perception Task, and 5 input neurons and 1 output neuron for the Number Comparison task. In addition, there are a variable number of

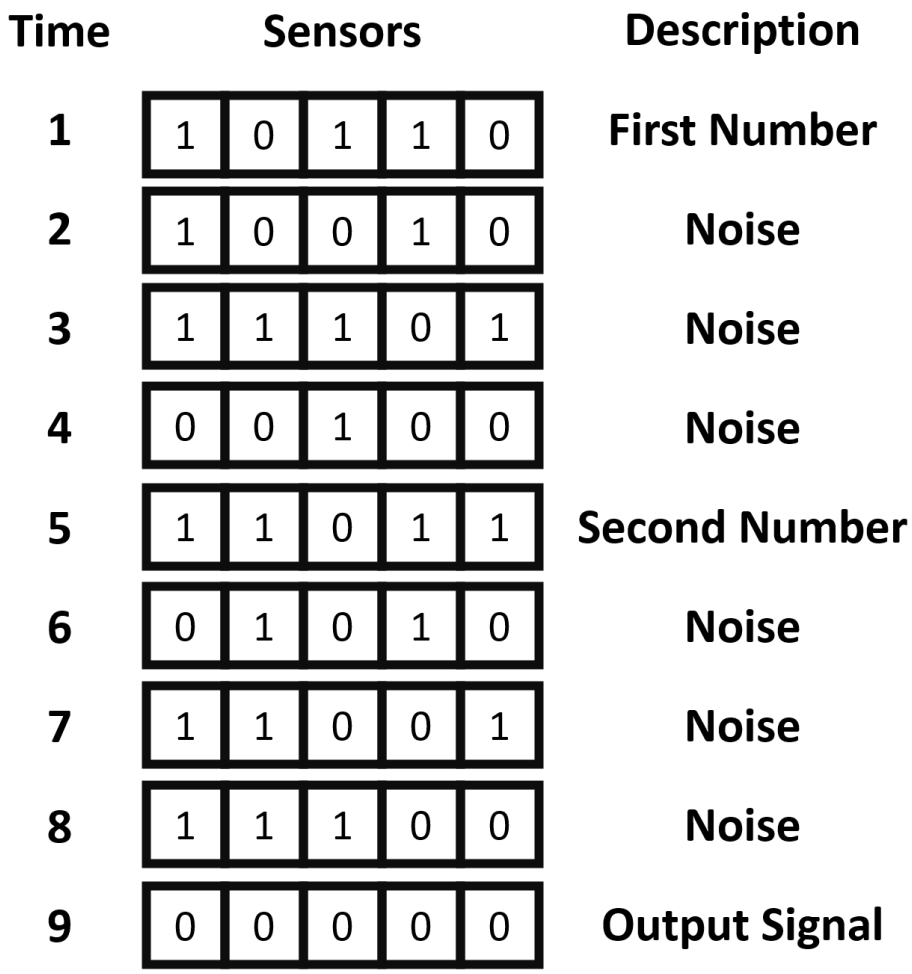

Figure 6.2: A single case of the Numerical Comparison Task. The agent receives a series of inputs over a period of 9 brain updates. The input values are shown in the squares under "Sensors", where each square represents a unique sensor input, and correlate with brain updates listed under "Time". The description of each sensor input is likewise listed under "Description". The agent receives a first number, here 3 (note that 3 of the sensors have a 1 as input), and then has 3 intermediate updates where noise is fed to each sensor with probability $p = 0.5$. The agent then receives a second number, here 4 (note likewise that 4 of the inputs are 1), and then a second period of 3 intermediate updates where noise is fed to the sensors with the same probability p. After the second set of noise inputs, the agent receives an output signal of all '0' inputs, and is expected to output a 1 because the second number is larger than the first.

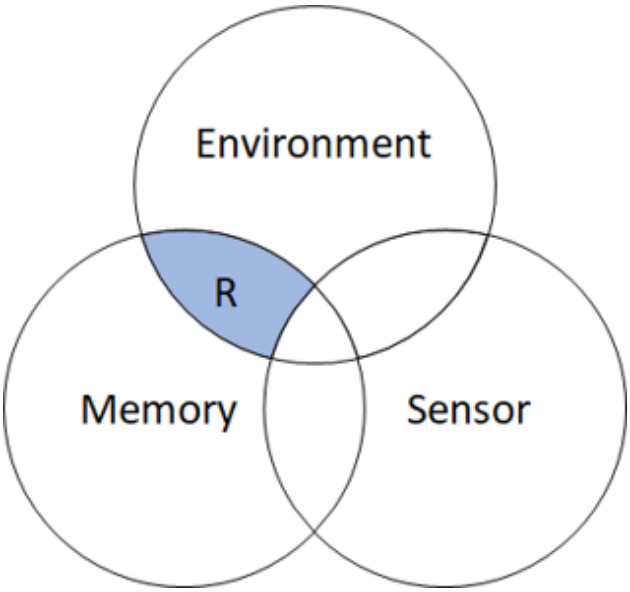

Figure 6.3: The information-theoretic Venn Diagram describing how $\mathcal{R}$ is calculated. $\mathcal{R}$ is defined as the information shared between the brain's memory and the salient features of the environment or task given the state of the sensors.

hidden nodes, here 8 for both tasks, which are only accessible internally and are recurrent to allow for the persistence of information and the formation of memory. These nodes are read-from or written-to by modular computational units that I simply call "gates". While a wide variety of gate types are available (e.g., single layer ANNs, Genetic Programming Functions, etc. - see [Hintze et al., 2017] for examples), here I only use the deterministic and probabilistic logic gates to connect neurons.

6.3.4 Mental Representations and Neuro-correlates

In order to quantify what the artificial cognitive system knows about its environment, I turn to information theory. Previous work has demonstrated the utility of two different measures: $\mathcal{R}$ or representation, which measures the amount of information about the world that the system has integrated, and $\mathcal{S}$ or smearedness, which measures the distribution of that information across the hidden state space or memory of the system.

6.3.4.1 Representations ($\mathcal{R}$)

[Marstaller et al., 2013] identify an information-theoretic measure, called $\mathcal{R}$, that identifies the amount of mental representation that an agent has stored about its environment. The mental representation is defined to be the information shared between the agent's recurrent memory and the environmental states given the information currently coming from the agent's sensors. $\mathcal{R}$ can be visualized in the information-theoretic Venn diagram, Fig. 6.3. In practice, I use Equation 6.2 to calculate $\mathcal{R}$, which was introduced in previous work (see Chapter 3).

$$\mathcal{R} = H(S, B) + H(S, E) - H(S) - H(E, B, S) \tag{6.2}$$

In Eq. 6.2, B refers to the hidden state or memory of the brain, E refers to the salient features of the environment or task, and S refers to the information coming from the sensors.

6.3.4.2 Smearedness ($\mathcal{S}$)

I am not only interested in the amount of representations measured, but also investigate the structure of said representations by considering an additional information-theoretic measure, smearedness or $\mathcal{S}$ [Hintze et al., 2018] to quantify the distribution of mental representations across the hidden states of the artificial cognitive system. First, observe that the memory state B can be deconstructed into individual neurons B_i, where $B = B_1 B_2 ... B_n$. Similarly, the environmental state E can be deconstructed into individual concepts E_j where $E = E_1 E_2 ... E_m$. $\mathcal{S}$ is calculated by first recording the 'atomic $\mathcal{R}$' or $\mathcal{R}_{ij}$, which is the information shared between *individual* brain states B_i and *individual* concepts E_j from the task, given the entire state of the sensors S. I then construct the matrix M where $M_{ij} = \mathcal{R}_{ij} = H(B_i : E_j|S)$. I then take a pairwise minimum is all concepts i and all nodes j and k to calculate $\mathcal{S}$ (see Equation 6.3).

$$\mathcal{S} = \mathcal{S}_C = \sum_i \sum_{j>k} min(M_{ji}, M_{ki}) \tag{6.3}$$

This can alternatively be done pairwise over nodes and concepts, however due to the high correlation between the two different calculations [Kirkpatrick and Hintze, 2019], I only use the first as $\mathcal{S}$ in the context of this paper.

6.3.5 Augmented Fitness Functions

For both the ACP and NCT, the typical fitness function used (see Equation 6.1) takes into account the correct (C) and incorrect (I) decisions that the agent makes to determine the score. The traditional method for encouraging mental representations in the fitness function, referred to as $\mathcal{R}$-augmentation, involves multiplying the original function by a normalized term including $\mathcal{R}$ [Schossau et al., 2015]. Schossau et al. find that the augmented function (Equation 6.4) results in the evolution of agents which have both higher levels of mental representation and better task performance.

$$W = 1.10^{(C-I)}(1 + \frac{\mathcal{R}}{\mathcal{R}_{max}}) \tag{6.4}$$

The augmentation function is modified with the addition of the $\mathcal{R}$ term so that the evolutionary process encourages higher values of $\mathcal{R}$ and simultaneously the formation of useful mental representations.

However, previous work [Hintze et al., 2018] shows that the structure of the mental representation may itself matter as much as the amount of representation present in the brain. Therefore, it seems plausible that I may also want to select for the structure of the representation. For example, I may choose to modify the augmented function to maximise the smearedness of the representations (see Equation 6.5).

$$W = 1.10^{(C-I)}(1 + \frac{\mathcal{S}}{\mathcal{S}_{max}}) \tag{6.5}$$

However, simple maximization may not work as well for the structure of the representations. Notably, [Hintze et al., 2018] suggests that a *lower* value of smearedness may work better for network robustness. Thus, I may try a matching augmentation method that encourages the evolution of *minimal* smearedness. The original augmented fitness function is structured so that the term containing $\mathcal{R}$ is evaluated to a floating point value between 1 and 2, so I shall do the same for our minimisation augmentation, with the reversed importance of smearedness (i.e., higher smearedness results in a lesser evolutionary benefit and vice-versa). This formulation results in a new fitness function, Equation 6.6.

$$W = 1.10^{(C-I)}(2 - \frac{\mathcal{S}}{\mathcal{S}_{max}}) \tag{6.6}$$

Alternatively, an absolute minimum of smearedness may not be ideal either, as some smearedness and consequently information distribution across the brain is likely necessary to perform computation and make decisions based on multiple concepts from the environment. Following the same constraints as for the minimal smearedness modification, I in its stead propose a new augmented fitness formula, Equation 6.7, that promotes the evolution of networks with smearedness measured close to a hypothetical ideal target value, T.

$$W = 1.10^{(C-I)}(2 - \frac{|\mathcal{S} - T|}{max\{T, \mathcal{S}_{max} - T\}}) \tag{6.7}$$

I structure the equation so that the difference between the observed value of $\mathcal{S}$ and T is minimized (i.e., the closer that T and $\mathcal{S}$ are, the larger the increase to W). I also maintain the normalization between 1 and 2 for the multiplicative term, as in Equations 6.4, 6.5, and 6.6. Here, I used target values of 2.0, 4.0, 6.0, 8.0, and 10.0 based on previously observed

values of smearedness.

6.3.6 Genetic Algorithm

All experiments were implemented using the MABE software framework [Bohm et al., 2017]. The circular genomes were restricted in size to be at least 2,000 sites and at most 20,000 sites long, where sites contained values in the range of 0 to 256. When I do not use MAP-Elites, I apply roulette-wheel selection to populations of 100 initially random agents with genomes of size 5,000. I apply mutations between generations using the default per-site point mutation rate of 0.005, insertion duplication rate of 0.00002 for chunks of size between 128 and 512, and a per-site deletion rate of 0.00002 for chunks with size between 128 and 512. After evolutionary optimization concluded, the line of descent (LOD) was recovered [Lenski et al., 2003], and subsequent analysis was conducted on agents from the LOD. I construct the LOD by maintaining a reference the parent of each organism as the agents are evolved, and subsequently finding the entire lineage of the best-performing agent in the population at the end of evolution. For each task, the experiments were run for 20,000 generations, with 100 replicate experiments conducted for each task and augmentation method.

6.3.7 MAP-Elites

The MAP-Elites algorithm [Mouret and Clune, 2015] works by maintaining a grid in multidimensional space (here, I will use $\mathcal{R}$ and $\mathcal{S}$ as the axes of a 2-dimensional space), where individual squares along the grid contain the highest-performing individual on a different measure (here the task fitness from Equation 6.1, W). After starting with an empty grid, 100 randomly generated Markov Brains are evaluated and placed on the 2-dimensional grid according to their respective values of $\mathcal{R}$ and $\mathcal{S}$ using bins of fixed width for each dimension. The intersection of one bin from the $\mathcal{R}$ axis (e.g., where $\mathcal{R}$ is between 1.4 and 1.6) and one bin from the $\mathcal{S}$ axis (e.g., where $\mathcal{S}$ is between 6.3 and 7.2) constitute a cell for that grid location, where the corresponding task score W is recorded. Afterwards, I take a randomly-selected

sample of 10 individuals from all of the agents in the grid, and then mutate them and evaluate them on the given task to generate scores for $\mathcal{R}$, $\mathcal{S}$, and W. Using these values for each new agent, I find the corresponding bin locations on the grid location, and if the new agent has a higher W than the agent in the cell, or the cell is empty, I place the new agent at that spot on the grid. I show an example of this process in Figure 6.4. I repeat this process for 200,000 updates, and run 100 replicate experiments. I averaged the values in the resulting heatmaps of the 100 replicate experiments, so that I take a more accurate estimation of the ideal values of $\mathcal{R}$ and $\mathcal{S}$. Based on the observation of values of $\mathcal{R}$ in previous work [Marstaller et al., 2013, Schossau et al., 2015, Kirkpatrick and Hintze, 2019] and in accordance with preliminary experiments (data not shown), I select a range of 0.0 to 2.0 for $\mathcal{R}$, with 10 equally distributed bins across those values to provide proper resolution without subdividing too far. As $\mathcal{S}$ has been shown to have a greater range of potential values (see Chapter 3 or [Hintze et al., 2018, Kirkpatrick and Hintze, 2019]), I use a similar selection methodology to set the bins for smearedness to be more numerous and across a slightly larger range - 20 bins equally distributed over the range from 0.0 to 18.0. Any values below the range specified for either dimension are grouped together in 1 bin, and any values above the range specified are also grouped together, in a separate bin.

6.4 Results and Discussion

Schossau et al. show that $\mathcal{R}$-augmentation can improve the performance of a genetic algorithm. Without $\mathcal{R}$-augmentation, mutations that improve a network's internal model often do not convey an immediate fitness advantage. When using $\mathcal{R}$-augmentation, those mutations are not neutral to evolution but instead contribute to fitness. Consequently, $\mathcal{R}$-augmentation accelerates genetic search [Schossau et al., 2015]. $\mathcal{R}$-augmentation assumes that a system with a high $\mathcal{R}$-value is in general better than a system with a lower $\mathcal{R}$-value. Is the same true for smearedness? Mutations that affect information smearing could be neutral and only

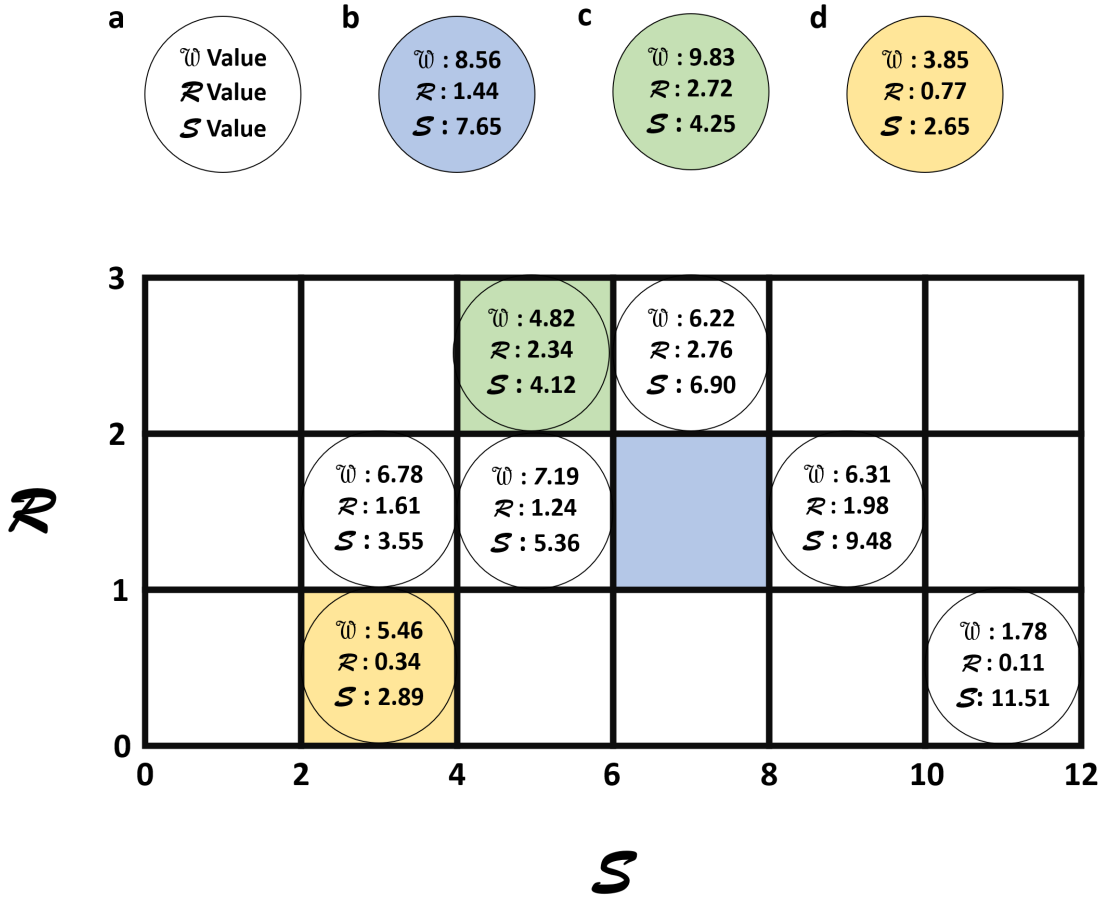

Figure 6.4: A simplified version of the MAP-Elites algorithm. Each organism is represented by the circle, and each square represents a bin for the algorithm. A prototypical organism is labeled (a), and demonstrates that each agent has a task performance score ω along with a measurement of the agent's $\mathcal{R}$ and $\mathcal{S}$. The grid is defined in terms of $\mathcal{S}$ on the x-axis and $\mathcal{R}$ on the y-axis. The bins for $\mathcal{R}$ are from 0 to 1, 1 to 2, and 2 to 3 while the bins for $\mathcal{S}$ are from 0 to 2, 2 to 4, 4 to 6, 6 to 8, 8 to 10, and 10 to 12. The algorithm will place the organism labelled (b) in the square with the same light blue shade based on the organism's $\mathcal{R}$ and $\mathcal{S}$ values. The organism will be successfully placed in the bin because there are no organisms in that bin previously. The algorithm will place the organism labelled (c) in the square with the same light green shade based on the organism's $\mathcal{R}$ and $\mathcal{S}$ values. The placement will succeed as the organism labelled (c) has a higher task performance than the organism already in the cell. The algorithm will attempt to place the organism labelled (d) in the bin of the same light yellow color based on the organism's $\mathcal{R}$ and $\mathcal{S}$ values. The placement will fail, however, because the organism that is already in the bin will remain because it has a higher task score, and the organism labelled (d) will be discarded.

later, in conjunction with other mutations, provide a fitness advantage. However, I do not know if smearedness should be maximized or minimized. Previous work [Hintze et al., 2018] finds that computational systems which have sparse connections to evolve low $\mathcal{S}$, while dense systems like fully connected neural networks evolve to have a high degree of smearedness. To explore if $\mathcal{S}$ should be maximized or minimized I compare both options for augmentation with a control experiment without augmentation, and an experiment using conventional $\mathcal{R}$-augmentation. I find that, as expected, $\mathcal{R}$-augmentation improves the performance of the genetic algorithm compared to the control on both tasks (see Figure 6.5, where the blue line represents the unaugmented control and the black line represents augmented data). At the same time, trying to augment by maximizing (using Eq. 6.5) or minimizing (using Eq. 6.6) smearedness results in worse performance than either $\mathcal{R}$-augmentation or the unaugmented control (see Figure 6.5, where the red line represents $\mathcal{S}$-maximization and the green line represents $\mathcal{S}$-minimization).

This confirms our intuition that intermediary levels of smearedness might be optimal as opposed to the extremes. To determine which value for smearedness is optimal, I repeated the experiment, but used an augmentation function that optimizes for a specific value of smearedness instead of just maximizing or minimizing it (see Equation 6.7). Specifically, the following target values were used: 2.0, 4.0, 6.0, 8.0, and 10.0.

I find several target values for smearedness that when used for augmentation perform equal to or better than the unaugmented control (see Figure 6.6) at varying points in evolutionary time. The target values of 4.0, 6.0, and 10.0 applied to the ACP task result in higher task performance than control at end of evolution, while the target value of 6.0 causes higher task performance than control at around 3000 generations into the evolutionary process, and equivalent performance to the unaugmented control at the end of evolution. I find that the exact target values of $\mathcal{S}$ that maximize performance seems to be task-dependent, and targeted augmentation only outperforms $\mathcal{R}$-augmentation on the ACP task, but not the NCT task. While this confirms that intermediary values can be used for augmentation towards a

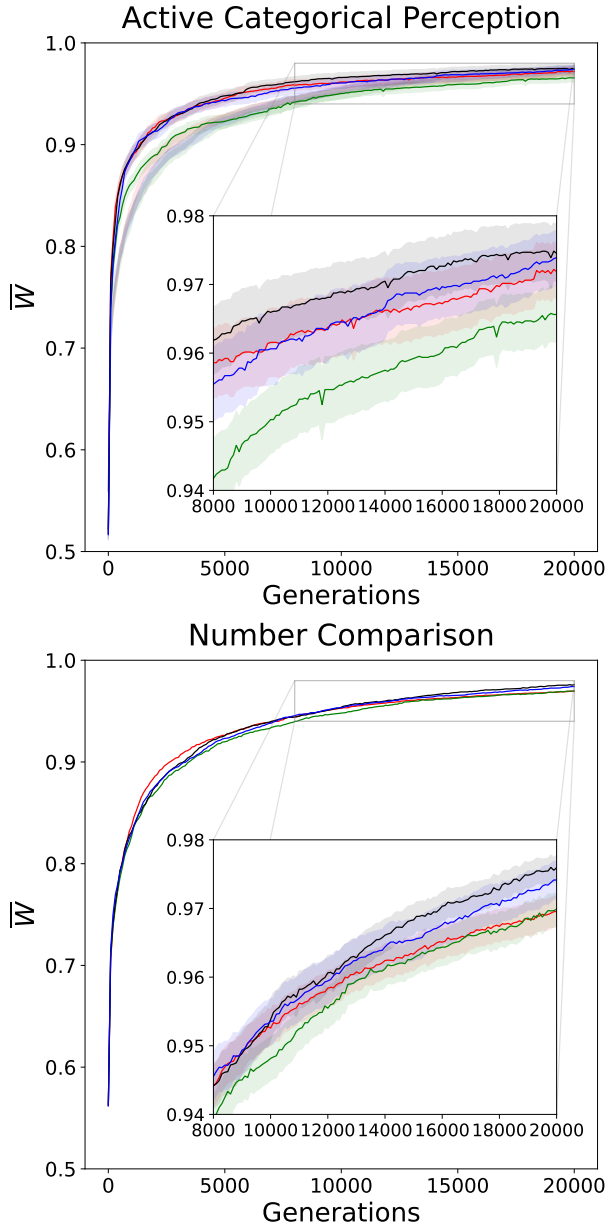

Figure 6.5: Average percent correct decisions over evolutionary time for the ACP and NCT tasks, where each line represents a different version of the fitness function used for that experiment. Each line is the average of 100 replicate runs using different random seeds. The blue line represents the original fitness function (Eq. 6.1). The black line represents $\mathcal{R}$-augmentation (Eq. 6.4). The red line represents $\mathcal{S}$-maximization (Eq. 6.5). The green line represents $\mathcal{S}$-minimization (Eq. 6.6). The shaded area of corresponding color around each line represents the standard error. The inset of the graph provides an expanded view of the indicated subset, to better distinguish the difference between conditions.

target value, such an approach is extremely wasteful with respect to computational resources: the performance gain is relatively small while many extra runs are needed to narrow in on the optimal value. This could mean that using augmentation towards a target value of $\mathcal{S}$ is only viable if the optimal value of $\mathcal{S}$ is known beforehand. It might be the case that there is an optimal value for smearing that is general for all computational systems, but from observations in other contexts I found no evidence for a consistent value of $\mathcal{S}$ across computational systems (data not shown).

As a simple sweep of values for smearedness may or may not produce an optimal value for the targeted evolution, and would require excessive computational time to verify, I elect to use a more comprehensive method, MAP-Elites, to identify a hypothetical ideal value for targeted evolution. Previous work [Mouret and Clune, 2015] shows that the MAP-Elite selection method explores fitness landscapes more effectively and potentially finds better solutions when other phenotypic or genotypic properties beyond task performance are measured. In our case, I use $\mathcal{R}$ and smearedness as the dimensions for the MAP-Elite space. Based on previous work [Kirkpatrick and Hintze, 2019] and preliminary experiments, there are 10 bins for $\mathcal{R}$ in the range from 0.0 to 2.0, leading to a bin width of 0.2. Due to increased variance in the previously observed [Kirkpatrick and Hintze, 2019, Hintze et al., 2018] values of $\mathcal{S}$, there are 20 bins for $\mathcal{S}$ over the range of 0.0 to 18.0, leading to a bin width of 0.9.

When evolving a populations of agents using MAP-Elites with $\mathcal{R}$ and smearedness as the dimensional variables, I find that as expected, optimal performing agents can be found for intermediary values of $\mathcal{S}$ (see Figure 6.7). However, surprisingly, those solutions that perform well having intermediary smearedness do not have maximal values of $\mathcal{R}$ at the same time. These results suggest that even though $\mathcal{R}$-augmentation works by maximizing $\mathcal{R}$, a maximized value of $\mathcal{R}$ might not even be the ideal goal for augmentation, and also that there is an interaction between the neuro-correlates $\mathcal{R}$ and $\mathcal{S}$.

I find that when using MAP-Elites, the best average performance over all replicate ex-

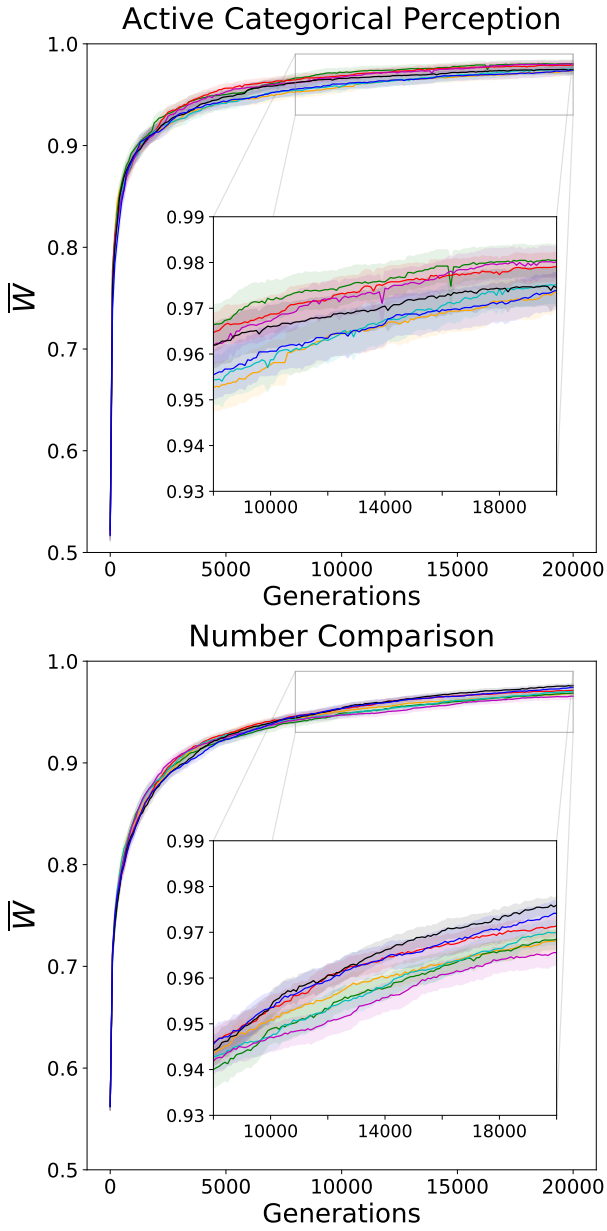

Figure 6.6: Average percent correct decisions over evolutionary time for the ACP and NCT tasks. Each line represents a different version of the fitness function used for that experiment and is the average of 100 replicate runs using different random seeds. As in Fig. 6.5, the black line represents $\mathcal{R}$-augmentation (Eq. 6.4), while the blue line represents the control. Each of the other differently colored lines represent $\mathcal{S}$-augmentation towards a target value (Eq. 6.7), for a range of different values. The correspondence of color to target values is: orange to 2.0, green to 4.0, red to 6.0, cyan to 8.0, and magenta to 10.0. The shaded area of corresponding color around each line represents the standard error. The inset of the graph provides an expanded view of the indicated subset, to exemplify the difference between conditions.

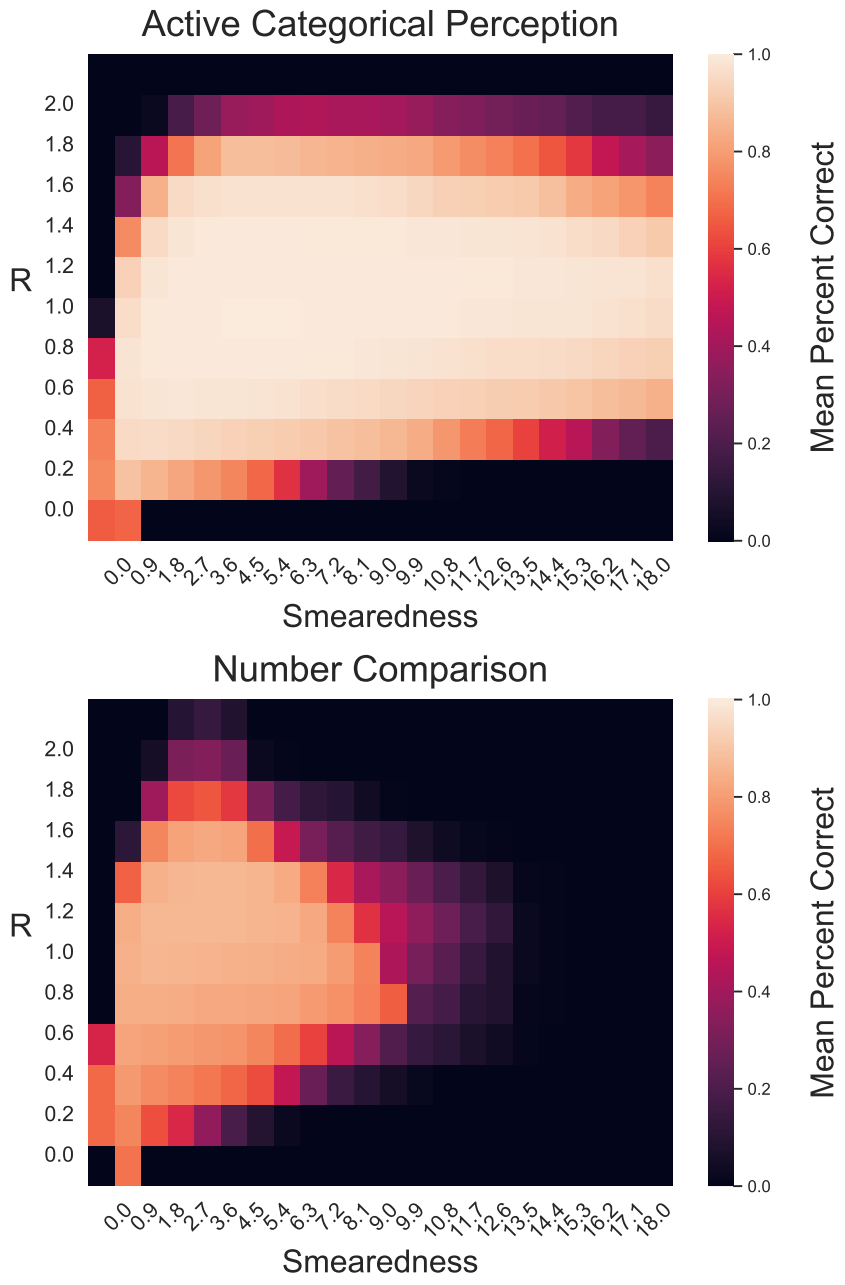

Figure 6.7: Heatmap of average MAP-Elite Results for the Active Categorical Perception Task and the Number Comparison Task. The top heatmap represents results from the ACP task, while the bottom heatmap represents results from the NCT task. Each rectangle represents the average value for a bin in MAP-Elites, for some subset of the range of $\mathcal{R}$ on the Y-axis and $\mathcal{S}$ on the X-axis. The value is the average percentage of correct decisions, with the highest average value occurring where $\mathcal{R}$ is between 0.8 and 1.0 for the ACP, and between 1.0 and 1.2 for the NCT task; and where $\mathcal{S}$ is between 3.6 and 4.5 for the ACP, and between 2.7 and 3.6 for the NCT task.

periment heatmaps for a specific combination of $\mathcal{S}$ and $\mathcal{R}$ in the ACP is 99.66% correct decisions and in the NCT is 86.66% correct decisions. This result can be directly compared to the previous optimizations using a regular GA or $\mathcal{R}$-augmentation (see Figure 6.5). In both cases 2,000,000 solutions were evaluated in total, and the average performance for the non MAP-Elite cases from an unaugmented GA were 97.41% for the ACP, and 97.39% for the NCT. Further, $\mathcal{R}$-augmentation with roulette selection improves the outcome performance by 0.5% or 1% for the NCT and ACP, respectively. While the absolute differences are not necessarily significant on their own, $\mathcal{R}$-augmentation finds the same performance as unaugmented evolution much earlier. When evaluated using the Mann-Whitney U test, I find that the performance at end of evolution for the agents evolved on the ACP task using MAP-Elites is significantly higher than the performance at end of evolution for the agents evolved on the ACP task using $\mathcal{R}$-augmentation ($p < 0.0001$). This shows that optimization using MAP-Elites using $\mathcal{R}$ and $\mathcal{S}$ can outperform $\mathcal{R}$-augmentation. Although I do not see the same effect when using MAP-Elites for the NCT task, it stands to reason that a more optimal setup of MAP-Elites may produce a better result for that task also. Notably, the heatmap for NCT in Fig. 6.7 shows a large amount of unused space where high values of $\mathcal{S}$ (e.g., greater than 10.0) are not reached. This indicates that the range of values chosen may be too large, reducing the selective pressure on the agents. This discrepancy in MAP-Elites performances on both tasks related to the range of $\mathcal{S}$ may also be linked to the nature of the tasks, as the ACP task requires more information integration than the NCT task, and as such can tolerate a larger range of values for $\mathcal{S}$. The identification of the exact reason for the discrepancy in performance between the two tasks, and more broadly the further refinement of the MAP-Elites method using neurocorrelates for generalized applications remains an open direction for future research.

6.5 Conclusion

Using neurocorrelates to improve the ability of a genetic algorithm to search for optimal solutions is a promising direction for current and future research. As $\mathcal{R}$-augmentation showed, a simple joint fitness function already improves performance. Naturally, one might think that adding more complex neurocorrelates would improve the augmentation process further. However, I have shown that maximizing or minimizing for $\mathcal{S}$ is not a rewarding endeavor. Although it appears that using the augmentation towards an optimal target neurocorrelate value for either $\mathcal{S}$ or $\mathcal{R}$ could produce a better outcome than either evolution alone or even the original $\mathcal{R}$-augmentation method, the lack of *a priori* knowledge about optimal target values limits the applicability of this method. MAP-Elites serves as a potential method to overcome these issues both in the process of identifying ideal target values and also as an alternate evolutionary scheme whose performance is on par with $\mathcal{R}$-augmentation. Measuring $\mathcal{R}$ and $\mathcal{S}$ during evolution and using it to define the "Map of Elites" not only finds which values for $\mathcal{R}$ and $\mathcal{S}$ correlate with finding optimal solutions, the method itself outperforms a standard genetic algorithm as well as $\mathcal{R}$-augmentation.

Obviously, the neurocorrelates $\mathcal{R}$ and $\mathcal{S}$ can only be used in systems that have recurrent hidden states, as only then can these values be properly measured. Successfully measuring these neurocorrelates depends on the ability of the experimenter to determine the salient features of the environment that should be reflected in mental representations. But given these limitations, using MAP-Elites in conjunction with these neurocorrelates is a promising approach. The future will tell as to what degree other neurocorrelates could be used. An example of an alternative neurocorrelate is information integration [Albantakis et al., 2014] which also does not necessarily linearly correlate with the performance of an agent [Joshi et al., 2013]. Another limitation is that here I only used two different simple computational tasks, active categorical perception and number comparison. While I see no reason that this method should not work for other tasks, it would be good for future research to use this

method on different tasks to confirm that it performs well. Additionally, although the focus of this paper was on Markov Brains, given that previous work (Chapter 3 or [Kirkpatrick and Hintze, 2019]) has shown that the information-theoretic tools used here work on other computational substrates as well, I can identify a number of open questions that seek to answer how we can best use this method to evolve other types of artificial cognitive systems (e.g., RNN, LSTM, or any other recurrent network architecture). To conclude, the methods presented here broadly allow for the shaping of representation quantity and structure, and are not limited to any one task or cognitive system.

6.6 Acknowledgements

The author would like to thank Chris Adami for his input and guidance on the paper.

This work was supported in part by Michigan State University through computational resources provided by the Institute for Cyber-Enabled Research. This material is based in part upon work supported by the National Science Foundation under Cooperative Agreement No. DBI-0939454.

Chapter 7

Conclusions and Future Work

Although this dissertation covers a number of different perspectives and questions, an underlying thread of understanding $\mathcal{R}$-augmentation runs through it all. Though the experimental questions may take the form of evaluating the effects of $\mathcal{R}$-augmentation on evolved networks, learning why $\mathcal{R}$-augmentation is able to create better performing cognitive systems, or understanding how to best apply $\mathcal{R}$-augmentation, each facet reflects the central theme of promoting task understanding in evolved artificial cognitive systems. In this chapter, I will give an overview of the major conclusions of this work as well as illuminate paths to future research.

When $\mathcal{R}$-augmentation was first introduced, some worried how the process might affect the properties of the evolved networks, including the potential for introduced drawbacks or flaws. Several of the chapters (e.g., Ch. 2 and Ch. 5) in this dissertation alleviate these concerns, demonstrating that beyond the additional computational cost, there are no noticeable drawbacks. I examined the ability of cognitive systems evolved with $\mathcal{R}$-augmentation to generalize to different tasks and to be robust to sensor noise. In some cases, Markov Brains evolved with $\mathcal{R}$-augmentation were able to generalize significantly better than those evolved without, but this result was not found in all experimental conditions. Also notable

was that Markov Brains evolved with $\mathcal{R}$-augmentation were able to evolve more effectively when affected by sensor noise than those evolved with a standard GA.

Another concern for $\mathcal{R}$-augmentation was that it had primarily been shown to work for Markov Brains, using only deterministic and probabilistic logic gates, but little was known about how the process would affect other network types. I examined the use of various other gate types, and found that $\mathcal{R}$-augmentation works to improve task performance regardless of the computational components selected for Markov Brains. Also, I investigated several new cognitive structures based on Genetic Programming (GP) trees, and found that although the effect of $\mathcal{R}$-augmentation was not pronounced in every case, generally cognitive systems evolved with $\mathcal{R}$-augmentation outperformed those evolved without. Notably, the systems that struggled the most with forming representations received the greatest performance benefit from the use of $\mathcal{R}$-augmentation.

While the advantage of $\mathcal{R}$-augmentation in terms of increased task performance was obvious, it was less clear how precisely this increased performance was achieved. Chapter 4 examines this question of how $\mathcal{R}$-augmentation works, with the conclusion that a change in evolutionary dynamics drives the difference in performance from networks created by an unaugmented GA. This change in dynamics includes selection of mutations that produce changes in $\mathcal{R}$ and task fitness simultaneously, as opposed to mutations that only favor one or the other. It may also be that $\mathcal{R}$-augmentation allows for the persistence of mutations that allow for useful mental representations, when they would otherwise fall victim to mutational drift, although I was not able to confirm this exact situation.

One of the notable results discovered in the course of this investigation of $\mathcal{R}$-augmentation is that the process allows for maintaining the same evolved performance while using fewer task evaluations than a standard GA. This can result in reduced total processing time even when factoring in the computational overhead necessary for the evaluation of $\mathcal{R}$. Fewer evaluations means reduced server time, more eco-friendly and efficient computation, and the opportunity

to pursue more complex tasks. The effect is seen here on a relatively simple task, and the reduction of trials would be even more beneficial on a computationally taxing task (i.e., a physics-based or 3D vision related task).

Finally, I show that optimizing for $\mathcal{R}$ can be achieved using other selection methods, using the MAP-Elites method as a demonstration. Instead of solely optimizing for a maximal measurement of mental representations, I investigate optimizing towards a hypothetical intermediary value for the distribution of representations, or smearedness. Such an approach may potentially provide better results, but the search necessary to find an ideal target value would likely be too costly to provide a net advantage. Instead, using the MAP-Elites optimizer to maintain diversity in both $\mathcal{R}$ and smearedness may produce a better result with the same number of agent evaluations. Both methods show promise in certain circumstances, but further refinement is required.

Overall, there are two main findings from this work. The first is that $\mathcal{R}$-augmentation works best for sparsely-connected networks that do not have a predetermined information flow (i.e., fixed connectivity), in tasks with well-defined states. That is, fully-connected networks like RNNs already have full information flow and potentially too much representation. Although 'too much representation' may seem like an odd concept, there is a potential for information overload. That is, if there are too many internal models that need to be selected from or combined together, it may be impossible for the network to come to a consensus and make the correct decision. In that case, $\mathcal{R}$-augmentation is not useful as those networks require a reduction of $\mathcal{R}$, whereas in sparse networks the representations need to be created, and $\mathcal{R}$-augmentation is more effective. The second finding is that in the cases seen thus far, $\mathcal{R}$-augmentation produces networks that perform better than or equal to the systems produced by unaugmented GAs. In tasks where the cost of evaluating $\mathcal{R}$ is small relative to the cost of task evaluation, there may be a large performance benefit to adding $\mathcal{R}$-augmentation to the GA with little overhead.

This work covers a lot of ground in evolutionary AI, but has the potential to drive impactful research in other fields, most notably Machine Learning and Deep Learning. Given the larger scale at which those fields operate, in terms of both tasks and network sizes, either the increase in performance offered by $\mathcal{R}$-augmentation or the potential reduction in trial cases could be useful to researchers and practitioners alike. Although perhaps not a direct analogue, modifying the loss function of a machine learning model to take into account the network's representation could result in an effect similar to evolutionary $\mathcal{R}$-augmentation. The performance increases from such a change could allow for more efficient training and less computation cost and run time. In deep learning there is an existing issue whereby the network overfits to a specific set of inputs and can no longer generalize to new data [Zhang et al., 2018]. Even worse, it appears as if deep learned neural networks are vulnerable to statistical regularities and can be susceptible to adversarial images which cause incorrect classifications [Jo and Bengio, 2018]. The ability of $\mathcal{R}$-augmented networks to evolve on a small subset of cases and still generalize to the whole set of data provides a framework for solving both of these issues. Namely, by forcing the deep learned network to maintain the representation of the whole task, it could also be trained on fewer data points while maintaining the ability to perform well on the whole. Using a rotating subset of training cases with $\mathcal{R}$-augmentation of the loss function, in a manner similar to that demonstrated in Chapter 5, could provide a method for avoiding overfitting to any one sample.

Another potential avenue of research falls in the realm of generalized intelligence. Optimizing for understanding on multiple tasks could lead to the evolution of a cognitive system that can perform new tasks previously unseen or with minimal training, and thus is more able to generalize and has a higher degree of adaptive cognition. This would be a form of meta- or multi-$\mathcal{R}$-augmentation. In order to evolve cognitive systems that can generalize, fitness landscapes encompassing more complex tasks and environments would be necessary. However, it would be in this complexity that we would finally be evolving artificial cognitive systems in the same way as their biological equivalents.

In summary, in this dissertation I discussed a range of topics related to $\mathcal{R}$-augmentation, creating a foundation for understanding how to encourage the evolution of mental representations. These topics include understanding the resulting changes or lack of changes that $\mathcal{R}$-augmentation causes in the networks created, investigating the evolutionary source of the improvement caused by $\mathcal{R}$-augmentation, and exploring other optimization methods and adaptations made possible by optimizing for mental representations. Although the results covered here are a good foundation for understanding $\mathcal{R}$-augmentation, there remain many outstanding questions for future research to explore.

BIBLIOGRAPHY

BIBLIOGRAPHY

[Albani et al., 2021] Albani, S., Ackles, A., Ofria, C., and Bohm, C. (2021). The comparative hybrid approach to investigate cognition across substrates. In *Pending - Artificial Life Conference Proceedings*.

[Albantakis et al., 2014] Albantakis, L., Hintze, A., Koch, C., Adami, C., and Tononi, G. (2014). Evolution of integrated causal structures in animats exposed to environments of increasing complexity. *PLoS computational biology*, 10(12):e1003966.

[Banzhaf et al., 1998] Banzhaf, W., Nordin, P., Keller, R. E., and Francone, F. D. (1998). Genetic programming: An introduction: On the automatic evolution of computer programs and its applications. dpunkt–verlag fur digitale technologie gbmh and morgan kaufmann publishers. *Inc., Heidelberg and San Francisco CA, resp.*

[Beer, 1996] Beer, R. (1996). Toward the evolution of dynamical neural networks for minimally cognitive behavior. In P. Maes et al., editor, *Proc. 4th Intern. Conf. on Simulation of Adaptive Behavior*, pages 421–429, Cambridge, MA. MIT Press.

[Beer, 2003] Beer, R. D. (2003). The dynamics of active categorical perception in an evolved model agent. *Adaptive Behavior*, 11(4):209–243.

[Bengio and Frasconi, 1995] Bengio, Y. and Frasconi, P. (1995). An input output hmm architecture. In *Advances in neural information processing systems*, pages 427–434.

[Bohm et al., 2017] Bohm, C., CG, N., and Hintze, A. (2017). MABE (modular agent based evolver): A framework for digital evolution research. *Proceedings of the European Conference of Artificial Life*.

[Brooks, 1991] Brooks, R. A. (1991). Intelligence without representation. *Artificial intelligence*, 47(1-3):139–159.

[Clune et al., 2013] Clune, J., Mouret, J.-B., and Lipson, H. (2013). The evolutionary origins of modularity. *Proc. R. Soc. B*, 280(1755):20122863.

[Clune et al., 2011] Clune, J., Stanley, K. O., Pennock, R. T., and Ofria, C. (2011). On the performance of indirect encoding across the continuum of regularity. *IEEE Transactions on Evolutionary Computation*, 15(3):346–367.

[Collins and Gentner, 1987] Collins, A. and Gentner, D. (1987). How people construct mental models. *Cultural models in language and thought*, 243:243–265.

[Dao et al., 2020] Dao, D. V., Ly, H.-B., Vu, H.-L. T., Le, T.-T., and Pham, B. T. (2020). Investigation and optimization of the c-ann structure in predicting the compressive strength

of foamed concrete. *Materials*, 13(5):1072.

[De Jong, 1975] De Jong, K. (1975). An analysis of the behavior of a class of genetic adaptive systems. *Ph. D. Thesis, University of Michigan*.

[Deb, 2001] Deb, K. (2001). *Multi-objective optimization using evolutionary algorithms*, volume 16. John Wiley & Sons.

[Deng, 2012] Deng, L. (2012). The mnist database of handwritten digit images for machine learning research [best of the web]. *IEEE Signal Processing Magazine*, 29(6):141–142.

[Edlund et al., 2011] 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 Comput Biol*, 7(10):e1002236.

[Floreano et al., 2008] Floreano, D., Dürr, P., and Mattiussi, C. (2008). Neuroevolution: from architectures to learning. *Evolutionary Intelligence*, 1(1):47–62.

[Fu, 2016] Fu, M. C. (2016). Alphago and monte carlo tree search: the simulation optimization perspective. In *2016 Winter Simulation Conference (WSC)*, pages 659–670. IEEE.

[Halford, 2014] Halford, G. S. (2014). *Children's understanding: The development of mental models*. Psychology Press.

[Handley, 1994] Handley, S. G. (1994). The automatic generations of plans for a mobile robot via genetic programming with automatically defined functions. *Advances in genetic programming*, 18:391–407.

[Helmuth et al., 2015] Helmuth, T., Spector, L., and Matheson, J. (2015). Solving uncompromising problems with lexicase selection. *IEEE Transactions on Evolutionary Computation*, 19(5):630–643.

[Hintze et al., 2017] Hintze, A., Edlund, J. A., Olson, R. S., Knoester, D. B., Schossau, J., Albantakis, L., Tehrani-Saleh, A., Kvam, P., Sheneman, L., Goldsby, H., Bohm, C., and Adami, C. (2017). Markov brains: A technical introduction. *arXiv preprint arXiv:1709.05601*.

[Hintze et al., 2018] Hintze, A., Kirkpatrick, D., and Adami, C. (2018). The structure of evolved representations across different substrates for artificial intelligence. In *Artificial Life Conference Proceedings*, pages 388–395. MIT Press.

[Hintze et al., 2019] Hintze, A., Schossau, J., and Bohm, C. (2019). The evolutionary buffet method. In *Genetic Programming Theory and Practice XVI*, pages 17–36. Springer.

[Hochreiter and Schmidhuber, 1997] Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, 9(8):1735–1780.

[Jo and Bengio, 2018] Jo, J. and Bengio, Y. (2018). Measuring the tendency of CNNs to learn surface stastistical regularities. arXiv:1711.11561.

[Johnson and Zhang, 2013] Johnson, R. and Zhang, T. (2013). Accelerating stochastic gradient descent using predictive variance reduction. *Advances in neural information processing systems*, 26:315–323.

[Johnson-Laird, 1980] Johnson-Laird, P. N. (1980). Mental models in cognitive science. *Cognitive science*, 4(1):71–115.

[Joshi et al., 2013] Joshi, N. J., Tononi, G., and Koch, C. (2013). The minimal complexity of adapting agents increases with fitness. *PLoS Comput Biol*, 9(7):e1003111.

[Kaur et al., 2018] Kaur, J., Kalra, A., and Sharma, D. (2018). Comparative survey of swarm intelligence optimization approaches for ann optimization. In *Intelligent Communication, Control and Devices*, pages 305–314. Springer.

[Kirkpatrick and Hintze, 2019] Kirkpatrick, D. and Hintze, A. (2019). The role of ambient noise in the evolution of robust mental representations in cognitive systems. In *Artificial Life Conference Proceedings*, pages 432–439. MIT Press.

[Konak et al., 2006] Konak, A., Coit, D. W., and Smith, A. E. (2006). Multi-objective optimization using genetic algorithms: A tutorial. *Reliability engineering & system safety*, 91(9):992–1007.

[Koza, 1994] Koza, J. R. (1994). Genetic programming as a means for programming computers by natural selection. *Statistics and computing*, 4(2):87–112.

[Koza and Rice, 1992] Koza, J. R. and Rice, J. P. (1992). Automatic programming of robots using genetic programming. In *AAAI*, volume 92, pages 194–207. Citeseer.

[Krizhevsky et al., 2009] Krizhevsky, A., Hinton, G., et al. (2009). Learning multiple layers of features from tiny images. *University of Toronto*.

[Langdon, 1995] Langdon, W. B. (1995). Evolving data structures with genetic programming. In *ICGA*, pages 295–302.

[LeCun et al., 1998] LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324.

[Lehman et al., 2018] Lehman, J., Clune, J., and Misevic, D. (2018). The surprising creativity of digital evolution. In *Artificial Life Conference Proceedings*, pages 55–56. MIT Press.

[Lehman and Stanley, 2008] Lehman, J. and Stanley, K. O. (2008). Exploiting open-endedness to solve problems through the search for novelty. In *ALIFE*, pages 329–336.

[Lenski et al., 2003] Lenski, R. E., Ofria, C., Pennock, R. T., and Adami, C. (2003). The evolutionary origin of complex features. *Nature*, 423:139–144.

[Liu and Deng, 2015] Liu, S. and Deng, W. (2015). Very deep convolutional neural network based image classification using small training sample size. In *2015 3rd IAPR Asian conference on pattern recognition (ACPR)*, pages 730–734. IEEE.

[Marstaller et al., 2013] Marstaller, L., Hintze, A., and Adami, C. (2013). The evolution of representation in simple cognitive networks. *Neural computation*, 25(8):2079–2107.

[McCulloch and Pitts, 1943] McCulloch, W. S. and Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133.

[Merritt and Brannon, 2013] Merritt, D. J. and Brannon, E. M. (2013). Nothing to it: Precursors to a zero concept in preschoolers. *Behavioural processes*, 93:91–97.

[Merritt et al., 2009] Merritt, D. J., Rugani, R., and Brannon, E. M. (2009). Empty sets as part of the numerical continuum: conceptual precursors to the zero concept in rhesus monkeys. *Journal of Experimental Psychology: General*, 138(2):258.

[Miller, 2011] Miller, J. F. (2011). Cartesian genetic programming. In *Cartesian Genetic Programming*, pages 17–34. Springer.

[Miller and Thomson, 2000] Miller, J. F. and Thomson, P. (2000). Cartesian genetic programming. In *European Conference on Genetic Programming*, pages 121–132. Springer.

[Minsky and Papert, 1969] Minsky, M. and Papert, S. (1969). An introduction to computational geometry. *Cambridge tiass., HIT*.

[Mohapatra et al., 2015] Mohapatra, R. K., Majhi, B., and Jena, S. K. (2015). Classification performance analysis of mnist dataset utilizing a multi-resolution technique. In *2015 International Conference on Computing, Communication and Security (ICCCS)*, pages 1–5. IEEE.

[Mouret and Clune, 2015] Mouret, J.-B. and Clune, J. (2015). Illuminating search spaces by mapping elites. *arXiv preprint arXiv:1504.04909*.

[Nguyen et al., 2015] Nguyen, A., Yosinski, J., and Clune, J. (2015). Deep neural networks are easily fooled: High confidence predictions for unrecognizable images. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 427–436.

[Nieder, 2018] Nieder, A. (2018). Honey bees zero in on the empty set. *Science*, 360(6393):1069–1070.

[Nordin, 1994] Nordin, P. (1994). A compiling genetic programming system that directly

manipulates the machine code. *Advances in genetic programming*, 1:311–331.

[Nordin and Banzhaf, 1995a] Nordin, P. and Banzhaf, W. (1995a). Genetic programming controlling a miniature robot. In *Working Notes for the AAAI Symposium on Genetic Programming*, volume 61, page 67. MIT, Cambridge, MA, USA, AAAI.

[Nordin and Banzhaf, 1995b] Nordin, P. and Banzhaf, W. (1995b). *A genetic programming system learning obstacle avoiding behavior and controlling a miniature robot in real time*. Univ., Systems Analysis Research Group.

[Nordin and Banzhaf, 1997] Nordin, P. and Banzhaf, W. (1997). Real time control of a khepera robot using genetic programming. *Control and Cybernetics*, 26:533–562.

[Olson et al., 2016] Olson, R., Adami, C., and Tehrani-Saleh, A. (2016). Flies as ship captains? digital evolution unravels selective pressures to avoid collision in drosophila. In *Proceedings of the Artificial Life Conference 2016 13*, pages 554–561. MIT Press.

[Pan and Yang, 2009] Pan, S. J. and Yang, Q. (2009). A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, 22(10):1345–1359.

[Parsopoulos and Vrahatis, 2002] Parsopoulos, K. E. and Vrahatis, M. N. (2002). Particle swarm optimization method in multiobjective problems. In *Proceedings of the 2002 ACM symposium on Applied computing*, pages 603–607.

[Phillips and Singer, 1997] Phillips, W. and Singer, W. (1997). In search of common foundations for cortical computation. *Behav Brain Sci*, 20:657–683.

[Pinker, 2013] Pinker, S. (2013). *Learnability and Cognition, new edition: The Acquisition of Argument Structure*. MIT press.

[Reynolds, 1993] Reynolds, C. W. (1993). An evolved, vision-based behavioral model of coordinated group motion. *From animals to animats*, 2:384–392.

[Reynolds, 1994] Reynolds, C. W. (1994). Evolution of obstacle avoidance behavior: using noise to promote robust solutions. *Advances in genetic programming*, 1:221–241.

[Russell and Norvig, 2016] Russell, S. J. and Norvig, P. (2016). *Artificial intelligence: a modern approach*. Malaysia; Pearson Education Limited,.

[Russell et al., 2003] Russell, S. J., Norvig, P., Canny, J. F., Malik, J. M., and Edwards, D. D. (2003). *Artificial intelligence: a modern approach*, volume 2-9. Prentice hall Upper Saddle River.

[Schossau et al., 2015] Schossau, J., Adami, C., and Hintze, A. (2015). Information-theoretic neuro-correlates boost evolution of cognitive systems. *Entropy*, 18(1):6.

[Spector and Robinson, 2002] Spector, L. and Robinson, A. (2002). Genetic programming and autoconstructive evolution with the push programming language. *Genetic Programming and Evolvable Machines*, 3(1):7–40.

[Srivastava et al., 2014] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958.

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

[Stanley and Miikkulainen, 2002] Stanley, K. O. and Miikkulainen, R. (2002). Evolving neural networks through augmenting topologies. *Evolutionary computation*, 10(2):99–127.

[Sun et al., 2019] Sun, S., Cao, Z., Zhu, H., and Zhao, J. (2019). A survey of optimization methods from a machine learning perspective. *IEEE transactions on cybernetics*, 50(8):3668–3681.

[Tehrani-Saleh et al., 2018] Tehrani-Saleh, A., LaBar, T., and Adami, C. (2018). Evolution leads to a diversity of motion-detection neuronal circuits. In *Artificial Life Conference Proceedings*, pages 625–632. MIT Press.

[Tehrani-Saleh et al., 2019] Tehrani-Saleh, A., McAuley, J. D., and Adami, C. (2019). Mechanism of perceived duration in artificial brains suggests new model of attentional entrainment. *bioRxiv*, page 870535.

[Teller, 1994] Teller, A. (1994). The evolution of mental models. *Advances in genetic programming*, pages 199–220.

[Thomas, 1996] Thomas, B. (1996). *Evolutionary algorithms in theory and practice*. Oxford University Press, New York.

[van Dartel et al., 2005] van Dartel, M., Sprinkhuizen-Kuyper, I., Postma, E., and van den Herik, J. (2005). Reactive agents and perceptual ambiguity. *Adaptive Behavior*, 13:227–42.

[Wagner, 2012] Wagner, A. (2012). The role of robustness in phenotypic adaptation and innovation. *Proceedings of the Royal Society B: Biological Sciences*, 279(1732):1249–1258.

[Wang et al., 2017] Wang, D.-J., Liu, F., and Jin, Y. (2017). A multi-objective evolutionary algorithm guided by directed search for dynamic scheduling. *Computers & Operations Research*, 79:279–290.

[Yao, 1999] Yao, X. (1999). Evolving artificial neural networks. *Proceedings of the IEEE*, 87(9):1423–1447.

[Zhang et al., 2018] Zhang, C., Vinyals, O., Munos, R., and Bengio, S. (2018). A study on overfitting in deep reinforcement learning. *arXiv preprint arXiv:1804.06893*.

[Zhou et al., 2011] Zhou, A., Qu, B.-Y., Li, H., Zhao, S.-Z., Suganthan, P. N., and Zhang, Q. (2011). Multiobjective evolutionary algorithms: A survey of the state of the art. *Swarm and Evolutionary Computation*, 1(1):32–49.