The Comparative Hybrid Approach to Investigate Cognition across Substrates

Sarah Albani[1,2,6], Acacia L. Ackles[3,5,6], Charles Ofria[4,5,6] and Clifford Bohm[3,6]

[1]Department of Neuroscience [2]Lyman Briggs College [3]Department of Integrative Biology [4]Department of Computer Science
[5]Program in Ecology, Evolution, and Biology [6]BEACON Center for the Study of Evolution in Action
Michigan State University, East Lansing, MI 48823
sarahalbani@gmail.com

Abstract

Understanding the structure and evolution of cognition is a topic of broad scientific interest. Computational substrates are ideal for conducting investigations into this topic because they can be incorporated in rapidly evolving Artificial Life systems and are easy to manipulate. However, design differences between currently existing digital systems make it difficult to identify which manipulations are responsible for broad patterns in evolved behavior. This is further confounded if we are trying to disentangle how multiple features interact. Here we systematically analyze components from two evolvable digital neural substrates (Recurrent Artificial Neural Networks (RNNs) and Markov brains) to develop a proof-of-concept for a comparative hybrid approach. We identified elements of the logic and memory storage architectures in each substrate, then altered and recombined properties of the original substrates to create hybrid substrates. In particular, we chose to investigate the differences between RNNs and Markov Brains relating to network sparsity, whether memory is discrete or continuous, and the basic logic operator in each substrate. We then tested the original substrates and the hybrids across a suite of distinct environments with different logic and memory requirements. While we observed trends across all three of the axes that we investigated, we identified discreteness of memory as an especially important determinant of performance across our test conditions. However, the specific effect of discretization varied by environment and whether the associated task relied on information integration. Our results demonstrate that the *comparative hybrid approach* can identify structural components that enable cognition and facilitate task performance across multiple computational structures.

Introduction

The concept of cognition is a subject of study in both neuroscience and artificial intelligence. Studying cognition can lead to a greater theoretical understanding of what allows cognitive processes to function, as well as an understanding of how to harness these processes for problem-solving.

Much of machine learning was inspired by the effective mechanisms found in natural systems, which may explain many of their functional similarities. However, while natural brains are effective at producing robust forms of general cognition, human-designed digital structures have not yet demonstrated general intelligence. On the one hand, it may be that we have not gained a sufficient understanding of biological intelligence to implement it in silico. On the other hand, perhaps modern computer systems simply lack some fundamental properties (such as massive parallelism) which make a direct implementation of a biological algorithm for intelligence infeasible. There are a number of disparate approaches to AI based different understandings of biological systems (e.g. ANN (Rosenblatt, 1958), HTM (Hawkins et al., 2011), spiking neural nets (Ghosh-Dastidar and Adeli, 2009)) or built up from mathematical or logical foundations (e.g. Avida (Ofria and Wilke, 2004), Markov Brains (Hintze et al., 2017), Signal GP (Lalejini and Ofria, 2018), Tangled Program Graphs (Kelly and Heywood, 2017)). A real problem in the field is an inability to directly compare these various systems and to draw concussions across the various approaches. This makes can make it difficult apply discoveries in one area of AI research to others and to relate research results from AI and neuroscience. Therefore, from an Artificial Life perspective, we want to expand beyond biomimetic approaches to producing machine intelligence and move towards a more holistic view that includes the underlying evolutionary processes that produced general cognition in nature.

Evolution took billions of years to produce agents that engage in intelligent behavior; we would prefer not to wait that long. While evolution in natural systems is driven by an underlying stochastic process, we are not restricted by the rate of random occurrences of beneficial mutations. We can take a more directed approach by systematically studying the evolution of cognition in digital systems. Ideally, we want to provide hands-on guidance to produce AI in a much shorter time, while simultaneously harnessing the creative and constructive potential of evolution to allow our digital systems to reach their full utility. To bootstrap this process, we must conduct a systematic study of potential evolutionary building blocks and underlying representations that are both computationally efficient and effective for evolving cognition.

Previous work has often compared whole neural archi-

tectures' performance on a particular task of interest. For example, algorithm performance has been studied in detail on tasks such as general classification (Williams et al., 2006; Singh et al., 2016; Binkhonain and Zhao, 2019), text recognition (Khan et al., 2019), disease prediction (Uddin et al., 2019), and ecological modeling (Crisci et al., 2012), to name a few. However, due to the numerous details related to design and parameterization of any particular structure, performance comparisons can not necessarily reveal why one structure outperforms another, nor investigate how their component parts may account for the results.

Some work, particularly in algorithm optimization, has dealt with combining components of multiple structures to create efficient computational hybrids. One such technique is the "Buffet Method", which allows a computational structure to access an array of different types of information processing sub-structures during the evolutionary process(Hintze et al., 2019) and shows that the proportion of sub-structures used in the evolved solutions is task dependent. Another method known as "Auto ML" is designed to evolve a substrate from basic mathematical principles (Real et al., 2020). While these methods have been successful in optimizing task performance, they rarely look "under the hood" at the exact variables responsible for cognitive success.

Our goal here is to demonstrate an approach for comparing cognitive substrates that allows us to identify which aspects of each substrate confer strengths in evolving solutions to cognitive processing tasks. Here, "substrate" refers to the virtual hardware underlying each digital representation—that is, the stuff from which each model of cognition is built. This concept of comparing two brains by testing hybrids based on these differences is what we have termed the *Comparative Hybrid Approach*. In this approach, we first identify the minimal number of differences between each substrate and then develop hybrid versions which represent intermediate forms. These are evaluated on different cognitive tasks in order to determine if some of the identified differences can account for performance variance. While our current focus is on only two such systems, a wider application of this approach will allow us to isolate properties of different types of components across more substrates and ultimately provide clearer guidance for designing new types of evolvable cognition.

This method is similar to knock out experiments in biology, where some aspect of a system is disabled and tests are conducted to identify changes to the system's response as a whole.

The conceptual backing for this method could be extended to more complex systems, from comparisons between more divergent digital structures to, simulations of accurate biologically-based models, or even actual biological systems (although this would require the ability to manipulate biological structures in a controlled manner).

In this work we compare Markov Brains and Recurrent Artificial Neural Networks (RNNs). In particular we identify differences in the logic that each brain has access to, how the logic units are connected (sparsity), and how memories are stored (discretization). We find that, while we observed performance differences on each axis of change, discretization in particular was the strongest indicator of task performance.

Methods

MABE: Modular Agent-Based Evolver

For the experiments in this work we used MABE, the Modular Agent Based Evolver framework (Bohm et al., 2017). We have configured MABE so that agents have a **genome** (a collection of heritable and mutable data) and a **brain** (a computational structure that receives input and generates output) specified by the genome. MABE manages the evolution of populations of agents by repeatedly evaluating agents in **worlds** (a fitness-bearing task) and using the results to select parents whose mutated offspring make up future generations. The advantage of the MABE framework is that the brains and worlds are completely modularized and therefore can be easily exchanged without altering other system properties, allowing for fast, direct comparisons. This "plug-and-play" feature makes MABE well suited to not only our current cross-brain and cross-world examination, but also to the proposed future extensions of this work.

Brains: RNNs and Markov Brains

In MABE, a brain is a process that converts a list of values T_0 to a new list of values T_1 (Fig 1A). We define the T_0 values as inputs (i.e. sensor readings or the state of the task) combined with the brain's memory and we define the T_1 values as outputs (i.e. values that determine behavior) combined with new memory values (which will be provided to the brain on the next update). Each time we want the brain to act (i.e. ask the brain "what will you do now?") we set the T_0 values, allow the brain to run its internal process, and read the resulting T_1 values.

For this work, we compare two well-studied computational structures: RNNs and Markov Brains. These structures can be implemented in a number of ways; we describe here our particular implementations. It is important to note that these particular implementations are designed not for speed of computation, but for ease of structural analysis.

In all of the experiments described in this work, both the RNNs and Markov Brains have eight memory values.

Recurrent Artificial Neural Networks: RNNs are a well-studied digital neural architecture; see Yu et al. (2019) for a review focused on RNNs and learning. Typically, RNNs are used in consort with back propagation or other machine learning methods; here, we are instead using neuroevolution. In our particular implementation, the internal structure

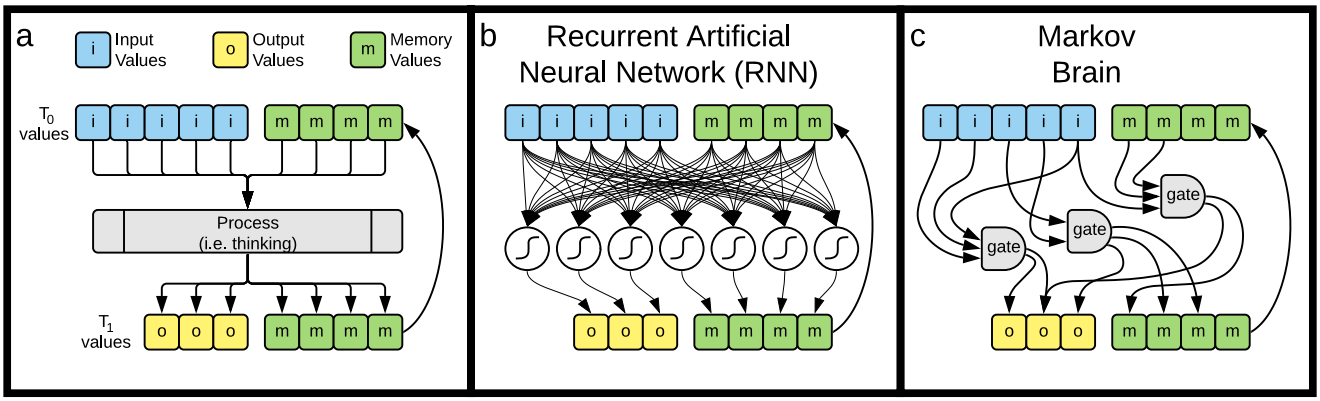

Figure 1: Schematics showing structures of A) a generic brain, B) an Recurrent Artificial Neural Network (RNN) and C) a Markov Brain.

of the RNN (Fig 1B) is a list of nodes, with one node for each T_1 value. Each node has a bias, and a weight for each T_0 value. When the RNN updates, each node adds its bias to the summation of the product of each T_0 value and its node-specific weight for that value, and applies $\tanh$ (a sigmoid function) to the result (which maps the value to the continuous range $[-1, 1]$). The resulting values are assigned to their respective output and memory values. The genome is used to determine the node weights and biases via a direct encoding.

Markov Brains: Markov Brains are a relatively new computational structure that have shown promise in studying evolution of learning (Hintze et al., 2017; Edlund et al., 2011; Sheneman and Hintze, 2017). The internal structure in the Markov Brains (fig:1C) consists of wires and gates. Markov Brains have been studied with a number of gate types, but here we used gates with binary lookup tables that have from 2 to 4 in-wires and 2 to 4 out-wires. In-wires connect T_0 values to gates and out-wires connect gates to T_1 values. When the Markov Brain updates, each gate uses its lookup table to convert input wire values to output values. Each T_1 value is the sum of the values provided by the output wires connected to it. An indirect encoding method is used to convert genome values into gates. Gates are defined by a "start codon" (a specific set of genome values) with the following values determining number of inputs, number of outputs, the wiring of inputs and outputs, and the lookup table.

Comparison of Brains Using the definition of the two brains, Markov Brains and RNNs, we can, in three steps, alter the description of the Markov Brain to that of the RNN, or visa versa.

Two of the alterations involve the internal processing and wiring of the brain. To alter the definition of a Markov Brain to that of an RNN we would first change the data processing unit from binary lookup table gates to summation and threshold nodes. Then, we would trade out the sparse wiring of the Markov brain for the fixed fully connected weighted wiring of the RNN. This identifies two obvious differences inherent to the internal architectures: the method of data processing and sparse vs dense connectivity.

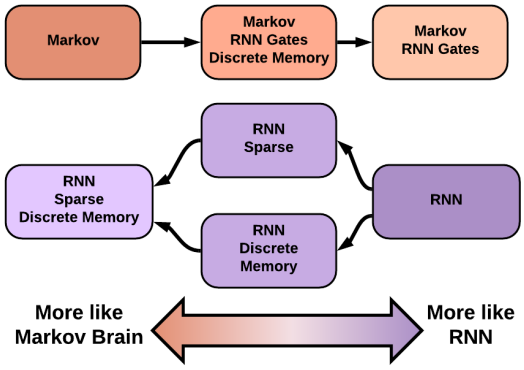

Figure 2: Schematic of hybrids. Orange (top) are Markov-based; purple (bottom) are RNN-based. Arrows indicate a single change away from the canonical brain structure.

The third required change relates not strictly to the internal architecture, but to the potential range of memory values. The binary lookup table gates in Markov brains are limited to binary values, while RNNs are able to operate on continuous value ranges. As a result, the 8 binary (i.e. discrete) memory values in the Markov Brain represent a much smaller set of possible states than the 8 continuous range memory values in the RNN. In fact, the third change—changing memory from discrete to continuous—is implicit and automatic, as the summation and threshold nodes operate on continuous value ranges.

Hybrid Brains We implemented a number of features in RNNs and Markov Brains to allow us to create "hybrid" brains based on each brain type (see Fig 2). These hybrids are representative of the architectural changes described above which theoretically transform one brain type into the other, and allow us to make those changes in a stepwise fashion, one change at a time.

For Markov brains, we implemented an RNN gate that can stand in place of the canonical lookup table gate. These gates have the same node summation and threshold properties as the RNNs. The wiring of the RNN gates is sparse and determined by the same genetic encoding as the lookup table gates with the exception that these gates have from 1 to 8 inputs and a single output. This replacement constitutes the change to the logical operator described above. To approximate the difference in the range of memory values, we added a setting so that memory is either bitted (i.e. mapped to binary such that values of 0 or less are set to 0 and values greater then 0 are set to 1) as in canonical Markov brains, or continuous as in canonical RNNs. These changes allow us to test three Markov-based brain variants: a canonical Markov Brain, a Markov-RNN hybrid with discrete memory (i.e. RNN gate with bitted memory), and a Markov-RNN hybrid with continuous memory (i.e. RNN gate without bitted memory).

In RNN brains, we directly varied the sparsity of their wiring and the discretization of memory. To implement sparsity, we biased the genome conversion so that wires with weight 0 were as common as wires with non-zero values. This meant that the inital population of randomly generated sparse brains would have on average half as many active connections as fully connected brains. As this ratio was only established at initialization, and not enforced thereafter, it could change as a result of evolution. To implement the discretization of memory, we added a bitting setting as in the Markov Brains. We tested four RNN variants: a canonical RNN, a sparse RNN (i.e. biased weights), a discretized RNN (i.e. bitted memory), and a sparse and discretized RNN (i.e. both biased weights and bitted memory).

Worlds

Since our work is motivated by understanding learning in the context of computational architectures, we tested each of our hybrid brains on three learning-based tasks with diverse characteristics. One task dealt primarily with short term memory (NBack), the second with simple information integration and lifetime memory (PathFollow), and the third with more complex information integration and lifetime memory (BlockCatch). We provide an in-depth analysis of the required logic and information integration required for each task and thoughts about how this relates to the results in the Discussion section titled Information Integration.

NBack World NBack world (Fig 3) presents a simple short-term memory task. In this task, agents are presented a bit stream, one bit at a time, and must respond with the last 8 bits they were provided. We tested each agent 10 times with 18 inputs presented per test. Only the outputs associated with the last 10 inputs were evaluated. Agents received a score based on the ratio of the correct outputs to the total number of outputs. NBack is based on a common cognitive task in neuropsychology to test working memory via neuroimaging, though its efficacy in human subjects is under review (Owen et al., 2005; Miller et al., 2009; Jaeggi et al., 2010).

PathFollow World In PathFollow world (Fig 3B), agents must learn to respond to a variable environmental clue. Agents exist on a grid and are able to sense symbols only on the space they are currently occupying. Agents must use those symbols to navigate a path defined by food and randomized turn signals. Agents can move forward and backwards, and turn left and right 45 degrees. Each location in the world contains a symbol that marks that location either as start, end, empty, food, turn signal 0, or turn signal 1. Agents start on the start location oriented towards the next step on the path. Visting a food location provides reward (used to determine reproductive success) and indicates a continuation of the path. If the agent steps off the path they receive a small penalty. The two turn signals indicate turns in the path, but these signals' meaning are reset every time an agent starts a new path. That is, signal 0 may indicate a left turn or a right turn, and signal 1 will indicate the opposite of the direction of signal 0. In order to perform well in Path Follow world, agents must form an association between the turn signals and their meaning by trial and error, and then must remember these associations for the remainder of that path. Agents are scored on their ability to reach the end of the path while visiting all locations and minimizing time off the path. PathFollow world is a simplified version of the task presented in Pontes et al. (2020); in the previous work, the turn signals were not restricted to a binary choice.

BlockCatch World BlockCatch world (Fig 3C) is a task that requires both lifetime memory and complex information integration. Agents are rewarded based on whether they catch or avoid a block that falls towards them (Marstaller et al., 2013). The world is a 20×20 grid, and agents are paddles that move left and right at the bottom of the world. Agents are 6 units wide and have 4 sensors that look up and can detect if there is an object directly above them, but not the distance to the object. The sensors are positioned such that the agent has a 2 unit wide blind spot over its center. Blocks of either size 2 or 3 begin at the top of the world and fall toward the agent one at a time. The blocks fall by moving horizontally one space, either left or right, and vertically down one space at a time. If a block is size 2 the

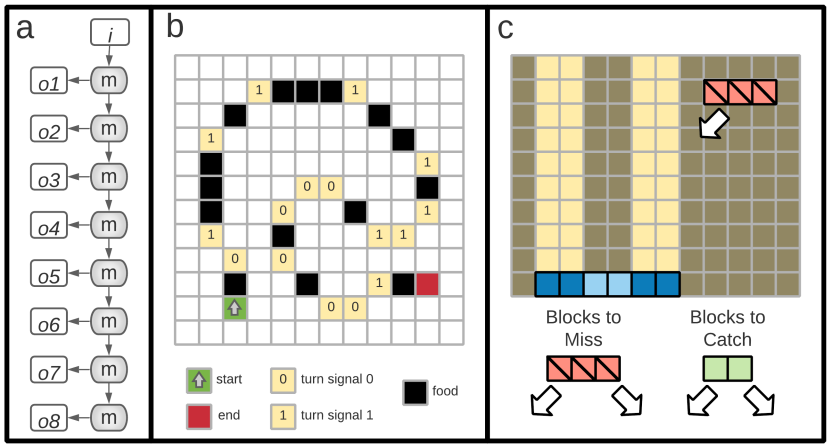

Figure 3: Depictions of the worlds on which the brains were evolved. (A) **NBack.** Visualization of the data flow required to perform the NBack 8 task of recalling the previous 8 given bits. Here i represents input, m represents memory, and $o1$ through $o8$ represent outputs. (B) **PathFollow.** Example map from the PathFollow task. Agents start on the green start square facing in the direction of the path. Black indicates path locations with food, where the agent must move forward and which the agent is rewarded for visiting. Yellow indicates locations with randomized turn signals which the agent must associate with either left of right turn. Red indicates the end of the path. Agents that reach the end of the path receive an additional reward if they have visited all "food" locations. (C) **BlockCatch.** Spatial layout of the BlockCatch task. Blue indicates the agent; dark blue indicates the location of the sensors. Bright yellow indicates the area visible to the sensors, while darker yellow indicates area that is not visible in the agent's current location. Green (blank) and red (hatched) depict blocks that should be caught or missed and arrows indicate possible lateral motion of blocks as they fall.

agent should "catch" the block (i.e. be positioned such that block intersects some part of the agent when the block has fallen to the level of the agent), and if a block is size 3 the agent should "miss" the block. Agents must integrate sensor input over time both to determine the block shape (to decide whether to catch or avoid) and the direction the block is moving (to correctly position themselves). They therefore must also remember the block size and direction once they have detected these features and behave accordingly. Agents are tested on every combination of block size and direction and receive a score that is the ratio of correct results over all tests.

Experiment Details Each described brain variant was evolved on each described world 100 times (i.e. 100 replicates of each experiment) for 200,000 generations. In all cases, we used populations of 100 agents and tournament selection (size 5). Agents' genomes were vectors of integer values in the range [0,255] with initial length 5,000. On reproduction, offsprings' genomes could be altered by insertion and deletion mutations of size 128 to 512 at a per site rate of 0.00002 and point mutations at a per site rate of 0.005. Genomes could not mutate to be shorter then 2,000 sites or longer than 20,000 sites.

Markov Brain genomes were seeded in the first generation of each experiment with six gate "start codons" for either lookup table gates or RNN gates depending on the brain variant.

All source code for the project, including analysis and visualization scripts, can be found at `https://doi.org/10.5281/zenodo.4765681`.

Results and Discussion

We present here results from an application of the Comparative Hybrid Approach where we compare Markov Brains and RNNs as a demonstration of the method's qualitative analytical power. By identifying architectural differences between brain types and creating intermediate "hybrids" of the two computational structures, we gain the ability to analyze how performance varied across hybrid structure and task. We are then able to examine whether our assumptions about the architectural differences separating the substrates map to the quantitative difference separating their performance.

Primary Results Analysis

We saw that Markov brains generally suffered significant performance loss when their logic lookup table gates were replaced with RNN-like threshold gates with continuous values. However, performance was stable or even increased when those threshold gates were used in consort with discrete memory. This indicates that discretization, rather than the logic lookup table structure, accounts, at least in part, for

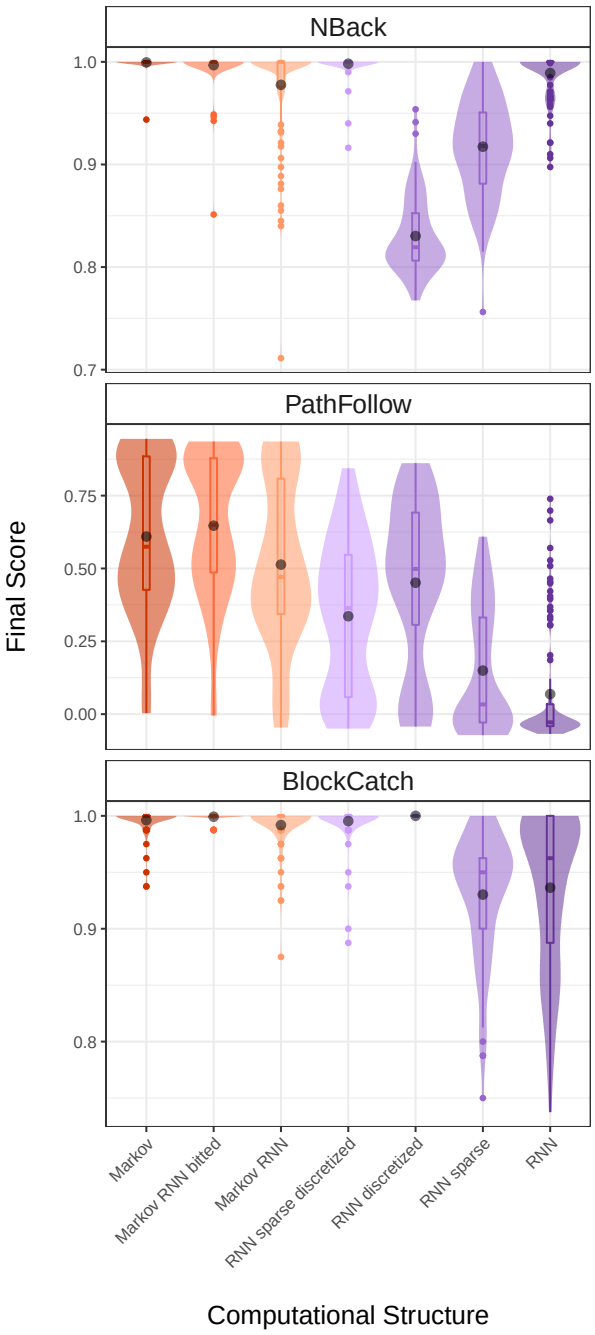

Figure 4: Performance of computational structures across tasks. Color hue indicates the brain variant used; color saturation indicates distance in number of changes from canonical structures, with more distant hybrids being less saturated. Note the varied scales on the y-axis, as we are interested here only in comparative performance within worlds rather than across worlds.

Significance: NBack

A	Markov RNN bitted	Markov RNN	RNN sparse/ discretized	RNN discretized	RNN sparse	RNN
Markov	1.000	0.943	1.000	*** <0.001	*** 0.002	0.999
Markov RNN bitted		0.971	1.000	*** <0.001	*** 0.003	1.000
Markov RNN			0.957	*** <0.001	0.06	0.998
RNN sparse/discretized				*** <0.001	** 0.002	1.000
RNN discretized					*** <0.001	*** <0.001
RNN sparse						** 0.01

Significance: PathFollow

B	Markov RNN bitted	Markov RNN	RNN sparse/ discretized	RNN discretized	RNN sparse	RNN
Markov	0.557	*** <0.001	*** <0.001	*** <0.001	*** <0.001	*** <0.001
Markov RNN bitted		*** <0.001	*** <0.001	*** <0.001	*** <0.001	*** <0.001
Markov RNN			*** <0.001	* 0.045	*** <0.001	*** <0.001
RNN sparse/discretized				*** <0.001	*** <0.001	*** <0.001
RNN discretized					*** <0.001	*** <0.001
RNN sparse						** 0.002

Significance: BlockCatch

C	Markov RNN bitted	Markov RNN	RNN sparse/ discretized	RNN discretized	RNN sparse	RNN
Markov	1.000	1.000	1.000	1.000	* 0.029	0.087
Markov RNN bitted		0.999	1.000	1.000	* 0.017	0.057
Markov RNN			1.000	1.000	0.055	0.146
RNN sparse/discretized				1.000	* 0.029	0.090
RNN discretized					* 0.015	0.051
RNN sparse						1.000

Table 1: Differences (p-value) between mean performance on tasks (A) NBack, (B) PathFollow, and (C) BlockCatch. Green cells indicate significant difference. P-values corrected using Tukey method for a family of 7 estimates. * $= p \leq 0.05$; ** $= p \leq 0.01$; *** $= p \leq 0.001$.

the high performance of Markov Brains on the tasks we investigated. The results are not as clear when we consider the RNN variants. While we can clearly see that the RNN variant which is both sparse *and* discretized out perform canonical RNNs in all three tasks, changes in independent task performance do not clearly indicate whether it is sparsity or discretization that is the primary factor—in fact, the results indicate that the benefits conferred by each feature are task dependent.

Overall, from these results, we tentatively conclude that discretizing outputs results in generally higher performance than biasing towards network sparsity. However, we note that all versions of Markov Brains tend to have higher sparsity compared to all versions of RNN, and so we can not discount that sparsity plays a significant role in explaining Markov Brains' generally higher performance.

Cross-Brain Examination: Discretization

Discretization Helps Performance on BlockCatch and PathFollow In two of the tasks, BlockCatch and PathFollow, discretizing the RNN memory resulted in significant performance gains, while sparsity of the network had little effect. In addition, the effect of combining sparsity and discretization resulted in performance that was lower than that of the RNNs that were only discretized. In all cases, hybrid Markov Brains (i.e. with RNN gates) display higher performance when discretized (i.e. bitted) in these same tasks. Moreover, we were surprised to see that in PathFollow, the Markov ANN bitted hybrids outperform the canonical Markov Brains.

Discretization or Sparsity Alone Hinder, But Together Help in RNN Hybrids Performance on NBack The advantages of discretization do not carry across all types of tasks as the results from NBack attest. The RNN NBack results are clearly different than the other two tasks in multiple ways; we see a performance gain *only* in the combined sparse-discretized network where as discretized-only and sparse-only RNN hybrids experienced performance *loss*.

Other Factors To Consider

Based on the contradictory results observed in relation to the NBack task, we turn to the examination of other specific brain features and task details for insights that might help to explain these results. The following are some conjectures on possible explanations for our observed results, and should be seen as theoretical.

Potential Influence of Encoding It is possible that only considering the difference between the RNN and Markov Brain architectures is insufficient to explain the differences in performance. Even though both structures are built from the same type of genomes with the same mutational properties, the process which is used to map the genomes to the two types of brains is quite different. RNNs use a direct encoding method where a contiguous region at the beginning of the genome is read and each position in the genome relates to a particular feature in the brain (a bias or a weight). In contrast, Markov Brains rely on biologically inspired "start codons" that mark coding regions. The RNN encoding means that the number of coding sites in the genome is constant, but because the number of start codons in a Markov Brains genome can vary, the number of coding sites can also vary. This results in a fixed number of effective mutations in RNNs but a variable number of effective mutations in Markov Brains. Because of these differences in encoding, the ways that brains are affected by mutation and the resulting evolutionary pathways may be very different. Further consideration of this is outside of the scope of this work, but we stress that the encoding methods could account for differences in observed performance and that future work should consider comparisons of encoding methods.

Information Integration

One factor that we believe could be critical to understanding these results and the results of our proposed method in general relate to the cognitive and memory requirements of the tasks being used. In two of the tasks, BlockCatch and PathFollow, we saw a gradient of performance across hybrids as we generally expected. The alterations of the RNNs caused them to perform more similarly to the Markov Brains (high performance), and the alterations of the Markov Brains caused them to perform more similarly to the RNNs (lower performance). However, this was not the case in the NBack task.

If we consider all three tasks from the standpoint of the cognitive abilities that each task requires, we can see that NBack only requires simple feed forward memory, while the other two tasks require long term memory and some level of information integration. Information integration is the ability to arrive at an answer that requires more than simple memory.

NBack can be solved by an agent that simply stores the incoming bit string in a "delay" buffer and delivers these stored values to the correct outputs over time. Each output relies only on a single prior input.

In PathFollow, agents must be able to integrate information over time. When the agent starts on a new path, they do not yet know which turn signal is associated with which turn. They must use trial and error to establish if they are in a "0 means left and 1 means right" path or a "1 means left and 0 means right" path. This requires that they make a guess when they arrive at the first turn signal. If they stay on the path then they "know" that they guessed correctly, but if they step off the path then they must not only remember that they guessed incorrectly, but also must navigate back onto the path. During this process (of getting back onto the path) their sensors will not provide them with any useful input,

so they must remember what part of the path recovery process they are executing. Finally, integration over sensors is needed. In order to detect a turn signal, agents must look at at least two inputs: the "turn signal" input, and either the "on turn" input or both the "on forward" and "off path inputs".

Finally, Block Catch requires that agents identify the size and direction of falling blocks. The identification of the left/right direction requires information integration (i.e. at least the integration of sensor states from consecutive time steps), while the identification of block size can be achieved by integration over time and sensor space or just over time.

From this it follows that the optimal logic and wiring for NBack should be simple and sparse. In fact there is no advantage in NBack to being able to integrate or even alter the input values. Arguably, a "pass though" gate which simply moves a single value from T to T+1 should be optimal. It is unclear why canonical RNNs perform well on NBack versus the discretized-only and sparse-only hybrids but we conjecture that these intermediate hybrids resulted in less navigable fitness landscapes.

PathFollow and Block Catch both require more complex models of memory and more complex methods of manipulating data than NBack. We believe that this accounts for the primary variance between the results of NBack and the results of PathFollow and Block Catch.

While this study was not designed to detect how information integration may impact optimal cognitive structures, our results indicate that such integration should be considered in future work.

Conclusion

We present these results as an illustrative example of the Comparative Hybrid Method and how it can be applied to analyze key components of digital brain architectures. By creating hybrids of existing examples of neuro architectures, we are able to test computational structures in a piece wise manner on a collection of tasks of varying complexity. The comparative performance of the unmodified and hybrid brains can be used to make inferences about how computational components and structure may relate to task success and ultimately to understanding fundamentally why the unmodified brains behave differently.

In practice, this technique can be extended to any two or more structures whose performance you are interested in comparing. All that is required is an intuition for what architectural differences separate the two computational structures, and an implementation of hybridized versions altering each of those components in turn. While such a task may be in some cases practically unwieldy, it will in many cases be simpler than the alternative of analytically examining the entire brain architecture at once and altering all possible parameters in order to identify key components.

In theory, the effects of the comparative hybrid model could extend into biological systems. As the ability to accurately simulate neuron models progresses, our method could be used across a continuum of neuron models and with the right advances in bio-engineering, the biological brain itself. Ultimately we believe that the Comparative Hybrid Method will allow us to single out key variables of cognition that could provide a path to a better understanding of intelligence, whether it be biological or computational.

Acknowledgements

This material is based in part upon work supported by a National Science Foundation Graduate Research Fellowship to A.A. and by the National Science Foundation under Cooperative Agreement No. DBI-0939454. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation. Michigan State University provided computational resources through the Institute for Cyber-Enabled Research. Michigan State University occupies the ancestral, traditional, and contemporary Lands of the Anishinaabeg–Three Fires Confederacy of Ojibwe, Odawa, and Potawatomi peoples. The University resides on Land ceded in the 1819 Treaty of Saginaw.

Author Contributions

C.B. conceived of the presented idea and study design. S.A. provided empirical backing for the theoretical design. C.B. developed components of the code base for the project. A.A. and C.B. performed computation and data collection. S.A., A.A., and C.B. performed data analysis and interpretation. S.A. drafted the initial manuscript. C.O. provided significant guidance on the manuscript. S.A., A.A., C.O., and C.B. revised and approved of the final manuscript.

References

Binkhonain, M. and Zhao, L. (2019). A review of machine learning algorithms for identification and classification of non-functional requirements. *Expert Systems with Applications: X*, 1.

Bohm, C., C G, N., and Hintze, A. (2017). MABE (Modular Agent Based Evolver): A framework for digital evolution research. In *Proceedings of the Fourteenth European Conference on Artificial Life*, pages 76–83. The MIT Press.

Crisci, C., Ghattas, B., and Perera, G. (2012). A review of supervised machine learning algorithms and their applications to ecological data. *Ecological Modelling*, 240:113–122.

Edlund, J. A., Chaumont, N., Hintze, A., Koch, C., Tononi, G., and Adami, C. (2011). Integrated information increases with fitness in the evolution of animats. *PLoS Computational Biology*, 7(10):1002236.

Ghosh-Dastidar, S. and Adeli, H. (2009). Spiking neural networks. *International journal of neural systems*, 19(04):295–308.

Hawkins, J., Ahmad, S., and Dubinsky, D. (2011). Hierarchical temporal memory (htm) whitepaper. *Numenta. Inc, Tech. Rep.*

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.

Hintze, A., Schossau, J., and Bohm, C. (2019). The Evolutionary Buffet Method. In Banzhaf, W., Spector, L., and Sheneman, L., editors, *Genetic Programming Theory and Practice XVI*, chapter 2, pages 17–36. Springer, Cham.

Jaeggi, S. M., Buschkuehl, M., Perrig, W. J., and Meier, B. (2010). The concurrent validity of the N-back task as a working memory measure. *Memory*, 18(4):394–412.

Kelly, S. and Heywood, M. I. (2017). Multi-task learning in atari video games with emergent tangled program graphs. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 195–202.

Khan, A., Baharudin, B., Hong Lee, L., and Khan, K. (2019). A Review of Machine Learning Algorithms for Text-Documents Classification. *Journal of Advances in Information Technology*, 1(1):4–20.

Lalejini, A. and Ofria, C. (2018). Evolving event-driven programs with signalgp. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 1135–1142.

Marstaller, L., Hintze, A., and Adami, C. (2013). The Evolution of Representation in Simple Cognitive Networks. *Neural Computation*, 25(8):2079–2107.

Miller, K. M., Price, C. C., Okun, M. S., Montijo, H., and Bowers, D. (2009). Is the N-Back Task a Valid Neuropsychological Measure for Assessing Working Memory? *Archives of Clinical Neuropsychology*, 24:711–717.

Ofria, C. and Wilke, C. O. (2004). Avida: A software platform for research in computational evolutionary biology. *Artificial life*, 10(2):191–229.

Owen, A. M., Mcmillan, K. M., Laird, A. R., and Bullmore, E. (2005). N-Back Working Memory Paradigm: A Meta-Analysis of Normative Functional Neuroimaging Studies. *Human Brain Mapping*, 25:46–59.

Pontes, A. C., Mobley, R. B., Ofria, C., Adami, C., and Dyer, F. C. (2020). The evolutionary origin of associative learning. *American Naturalist*, 195(1):E1–E19.

Real, E., Liang, C., So, D. R., and Le, Q. V. (2020). Automl-zero: Evolving machine learning algorithms from scratch. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119, pages 8007–8019.

Rosenblatt, F. (1958). The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386.

Sheneman, L. and Hintze, A. (2017). Evolving autonomous learning in cognitive networks OPEN. *Scientific Reports*, 7:16712.

Singh, A., Thakur, N., and Sharma, A. (2016). A review of supervised machine learning algorithms. In *Proceedings of the 10th INDIACom; 2016 3rd International Conference on Computing for Sustainable Global Development, INDIACom 2016*, pages 1310–1315. Bharati Vidyapeeth, New Delhi as the Organizer of INDIACom - 2016.

Uddin, S., Khan, A., Hossain, M. E., and Moni, M. A. (2019). Comparing different supervised machine learning algorithms for disease prediction. *BMC Medical Informatics and Decision Making*, 19(1):1–16.

Williams, N., Zander, S., and Armitage, G. (2006). A preliminary performance comparison of five machine learning algorithms for practical IP traffic flow classification. *Computer Communication Review*, 36(5):7–15.

Yu, Y., Si, X., Hu, C., and Zhang, J. (2019). A Review of Recurrent Neural Networks: LSTM Cells and Network Architectures. *Neural Computation*, 31:1235–1270.