

ABSTRACT

Title of Dissertation: NEUROCOMPUTATIONAL
METHODS FOR AUTONOMOUS
COGNITIVE CONTROL
Jared Sylvester, Doctor of Philosophy, 2014

Dissertation directed by: James Reggia,
Department of Computer Science

Artificial Intelligence can be divided between symbolic and sub-symbolic methods, with neural networks making up a majority of the latter. Symbolic systems have the advantage when capabilities such as deduction and planning are required, while sub-symbolic ones are preferable for tasks requiring skills such as perception and generalization. One of the domains in which neural approaches tend to fare poorly is cognitive control: maintaining short-term memory, inhibiting distractions, and shifting attention. Our own biological neural networks are more than capable of these sorts of executive functions, but artificial neural networks struggle with them. This work explores the gap between the cognitive control that is possible with both symbolic AI systems and biological neural networks, but not with artificial neural networks. To do so, I identify a set of general-purpose, regional-level functions and interactions that are useful for cognitive control in large-scale neural architectures. My approach has three main pillars: a region-and-pathway architecture inspired by

the human cerebral cortex and biologically-plausible Hebbian learning, neural regions that each serve as an attractor network able to learn sequences, and neural regions that not only learn to exchange information but also to modulate the functions of other regions. The resultant networks have behaviors based on their own memory contents rather than exclusively on their structure. Because they learn not just memories of the environment but also procedures for tasks, it is possible to “program” these neural networks with the desired behaviors.

This research makes four primary contributions. First, the extension of Hopfield-like attractor networks from processing only fixed-point attractors to processing sequential ones. This is accomplished via the introduction of temporally asymmetric weights to Hopfield-like networks, a novel technique that I developed. Second, the combination of several such networks to create models capable of autonomously directing their own performance of cognitive control tasks. By learning procedural memories for a task they can perform in ways that match those of human subjects in key respects. Third, the extension of this approach to spatial domains, binding together visuospatial data to perform a complex memory task at the same level observed in humans and a comparable symbolic model. Finally, these new memories and learning procedures are integrated so that models can respond to feedback from the environment. This enables them to improve as they gain experience by refining their own internal representations of their instructions. These results establish that the use of

regional networks, sequential attractor dynamics, and gated connections provide an effective way to accomplish the difficult task of neurally-based cognitive control.

NEUROCOMPUTATIONAL METHODS FOR AUTONOMOUS COGNITIVE CONTROL

by

Jared Sylvester

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2014

Advisory Committee:

Professor James Reggia, Chair

Professor William Rand

Professor Donald Perlis

Professor Jennifer Golbeck

Professor Michael Dougherty, Dean's Representative

© Jared Sylvester 2014

For L.P.

*Science is what we understand well enough to explain to a computer.
Art is everything else we do.*

Donald Knuth
FORWARD TO PETKOVSEK,
WILF & ZEILBERGER, 'A=B'

I have look'd on Worlds far distant, their Beauty how pitiless.

Thomas Pynchon
'MASON & DIXON'

Contents

List of Tables	vi
List of Figures	vii
1 Introduction and Rationale	1
1.1 The GALIS Framework	4
1.2 Goals and Aims	7
1.3 Overview	9
2 Background	11
2.1 The Nature of Human Working Memory	11
2.2 Past Neural Network Models of Working Memory	16
2.3 Past Neural Models for Cognitive Control	19
2.4 Past Neural Networks for Sequence Processing	34
2.5 Fast Weights & Gating	39
3 Associative Memories & Temporally Asymmetric Weights	48
3.1 Introduction	48
3.2 Methods	51
3.2.1 Model Description	51
3.2.2 Measuring Recall	55
3.2.3 Human Behavioral Data	57
3.3 Results	58
3.4 Discussion	62
4 Learning Instruction Sequences for Control	65
4.1 Introduction	65
4.2 The ‘Store & Recognize’ Task	68
4.2.1 Model Overview	71
4.2.2 Model Operation	78
4.2.3 Control Module Operation	79

4.2.4	Control Module Architecture	84
4.2.5	Results	92
4.3	The n -Back Task	95
4.3.1	Methods	97
4.3.2	Model Details	110
4.3.3	Results	123
4.4	Discussion	133
4.4.1	Testing the GALIS Hypothesis	133
4.4.2	GALIS as a general purpose framework	135
5	Visuospatial Processing and Binding	138
5.1	Introduction	138
5.2	Methods	139
5.2.1	Visual System	141
5.2.2	Executive System	145
5.2.3	Motor System	154
5.2.4	Experimental Methods	155
5.3	Further Details of the Card Matching Model	157
5.3.1	Location Layer	157
5.3.2	Object Module	159
5.3.3	Working Memory	160
5.3.4	Conflict Module	161
5.4	Results	162
5.5	Discussion	172
6	Learning, Improvement & Task Switching	176
6.1	Introduction	176
6.2	Methods	178
6.2.1	The Wisconsin Card Sort Test	178
6.2.2	Model Overview	180
6.2.3	Information Representation	181
6.2.4	Visual System	183
6.2.5	Working Memory	186
6.2.6	Controller	188
6.3	Results	194
6.4	Discussion	199
7	Discussion	202
7.1	Significance and Summary	202
7.2	Contributions	205
7.3	Limitations and Future Work	208
	Bibliography	212

List of Tables

3.1	Capacity of working memory model	60
3.2	Peak-to-peak transitions in correct order	60
4.1	Instruction sequences for Store/Recognize	80
4.2	Sample inputs & outputs for Store/Recognize accuracy tests	92
4.3	Sample inputs & outputs for Store/Recognize capacity tests	94
4.4	Instruction sequences for n -Back	109
5.1	Instruction sequences for Card Matching	151
6.1	Types of learning used for the WCST model	178
6.2	WCST input features and values	179
6.3	Instruction sequences for WCST	190

List of Figures

3.1	Example stimuli for the asymmetric weight sequential memory . . .	51
3.2	Sequential associative memory architecture	52
3.3	Ordering of WM recall with and without asymmetric weights . . .	59
3.4	Working memory model's recall rate by position	60
3.5	Comparison of WM model recall with human subjects	61
4.1	A memory divided into two conceptual partitions	68
4.2	Sample visual stimuli for Store/Recognize	69
4.3	Top-level architecture for the Store/Recognize task	70
4.4	Compare module for Store/Recognize model	73
4.5	Sequence of operations to add an item to working memory	81
4.6	Biasing attractor space to determine if a stimulus is in memory . .	83
4.7	Control module for Store/Recognize	85
4.8	Accuracy on the Store/Recognize task	93
4.9	Working memory capacity on the Store/Recognize task	94
4.10	Effect of decay on WM in the Store/Recognize model	96
4.11	Top-level architecture for n -Back	98
4.12	Control module for n -Back, simple version	101
4.13	Sequence of operations to process one stimulus for 3-back	108
4.14	Control module for n -Back	116
4.15	Comparison of model and human accuracies on n -Back	125
4.16	Model and human response times on n -Back	127
4.17	Accuracy when changing the value of n during a trial	128
4.18	Sources of error on n -Back	132
5.1	Visual environment for the Card Matching task	141
5.2	Model architecture for the Card Matching task	142
5.3	Model visual system for the Card Matching task	143
5.4	Model executive system architecture for the Card Matching task .	147
5.5	Detail of the conflict module architecture	162
5.6	Symbolic model performance on Card Matching	163

5.7	Performance of humans, GALIS & symbolic model	165
5.8	Accuracy distribution for GALIS when matching 12 & 16 cards . .	166
5.9	Accuracy distribution for humans, GALIS & symbolic model ($n=16$)	168
5.10	Visually-weighted regression of Card Matching performance	170
6.1	Visual field for Wisconsin Card Sort Test	179
6.2	Model architecture for WCST	181
6.3	Example object layer Self-Organizing Map	185
6.4	Schematic representation of WM contents for WCST	188
6.5	Effect of instruction refinement on performance	196
6.6	Performance distribution with & without instruction refinement .	197
6.7	WCST performance when trained with extraneous instructions . .	198
6.8	Instruction execution counts before and after refinement	199

Introduction and Rationale

“Cognitive control” is an umbrella term for those executive cognitive systems that manage other cognitive processes, such as working memory, planning, attention, inhibition, and action selection. Building neural architectures capable of modeling cognitive control processes is increasingly recognized as an important research direction for several reasons (Roy, 2008). First, improving the performance of neural networks in domains at which symbolic systems currently dominate will allow the strengths of neural networks, such as incremental learning and pattern recognition, to be leveraged (Omlin and Giles, 2000). Further, the capability to perform higher level executive functions like goal formation, planning and selective attention will make neural networks more autonomous, and hence more useful, in the future. Finally, neural models of human cognition may ultimately provide insight into human cognition and its neurobiological basis.

While these potential benefits are evident, developing such architectures has proven to be surprisingly challenging. Neural systems currently excel at problems that are limited to their strengths but they often struggle with problems requiring executive behaviors such as representing the goals and

rules of a task or constructing and carrying out procedures (Marcus, 2001). In contrast, symbolic AI systems face little difficulty with incorporating executive behaviors (Simen et al., 2010), due to the ease with which they can bind variables, create data structures, and perform global computations. This divergence in ability is particularly odd since biological neural systems do not experience the same difficulty that artificial neural networks do. For instance a person can typically play a novel card game merely after hearing the rules described, but a neural network might have to witness the game being played thousands of times before it can play it on its own. Why are cognitive control functions such as focusing on a goal state so easy for symbolic systems and living beings, but so difficult for artificial neural networks? It stands to reason that it should be just as possible for neural networks to perform these tasks as it is for the biological counterparts from which they draw inspiration.

There has been increasing interest during recent years regarding biologically-inspired computation that addresses these issues. Rather than just using neural networks as tools for applications at which they excel (character recognition, system control, etc.), many researchers are looking to understand the brain's computation from the bottom up, leveraging the link between neural AI systems and the brain. Examples of this interest are recent conferences (BICA, AGI, etc.), and research programs such as those as DARPA and IARPA. The growing interest in biologically-inspired computation has led to, among other things, the development of pioneering neural models that explicitly incorporate aspects of cognitive control, such as for managing working memory (O'Reilly and Frank, 2006) and for planning solutions to the Towers of London problem (Dehaene and Changeux, 1997, 2000; Samadi et al., 2008).

However, many such neural models are hard-wired for the particular task for which they are designed (Stewart et al., 2010), connection strengths are often set by hand without a learning procedure, and local conjunctive encodings are often used (e.g., Frank et al., 2001; Riachi et al., 2009; Varma and Just, 2006) specifying the exact sets of possible inputs and outputs and making adaptation to other situations, contexts or environments tricky. This specialization can make neural network models of cognitive control difficult to build, because each model requires not only parameter tuning and other human supervision, but often construction from the ground up. Even small changes in the task specifications can require large modifications to the architecture. For instance, the model for solving the Towers of London problem in Dehaene and Changeux (1997), while capable of an impressive amount of planning for a neural network, is incapable of solving the very similar Towers of Hanoi problem, or even of solving Towers of London using a method other than the greedy, depth-first search it has been constructed to execute. What would be helpful is the development of a general purpose, adaptive approach that, building on the successes of past specific implementations of cognitive control mechanisms, can be used for a broad range of applications.

This dissertation presents an approach to building neural models of cognitive control called GALIS, for “Gated Attractors Learning Instruction Sequences.” GALIS is intended to be a general-purpose, adaptive neurocomputational architecture that *learns* how to perform tasks, including tasks that themselves involve learning, within which models for specific tasks can be instantiated. The goal is not to provide a veridical model of the neurobiology underlying human cognitive control, but rather to take inspiration from the large-scale organization of the cerebral cortex to create a general computational framework

that can be used effectively to create a broad range of neural architectures for specific tasks. Similarly, while I will refer to GALIS as a cognitive model in the following, no claim is being made here that the GALIS framework is an accurate model of human cognitive algorithms, although we will see that at times its performance can align well with human data and can also make testable, falsifiable predictions (e.g., see the results related to the n -Back task in Section 4.3). All that is claimed is that the approach introduced here—i.e., gated transient-attractor networks—provides a substantial contribution to neurocomputational methods for executive control.

1.1 THE GALIS FRAMEWORK

While GALIS is not intended to be a neuroanatomical/physiological model of the brain circuitry underlying cognitive control, it is strongly inspired by contemporary views of the organization and functionality of primate cerebral cortex. Specifically, GALIS is derived from three main hypotheses about how cerebral cortex directs working memory, as follows.

The first hypothesis is that the cerebral cortex is organized as a distributed network of interacting cortical regions. Such a hypothesis is supported by a broad range of scientific evidence (Bressler and Menon, 2010; Sporns, 2011; van Essen et al., 1992). The implication for GALIS is that all aspects of working memory contents, both static information that captures task-specific details and dynamic procedures for performing a task, are stored within a network of model regions. In other words, model cortical regions must learn not only the “facts” about a specific instance of a task (as in most neural network systems), but also the procedure or “software” that is needed to perform that

task. Thus while GALIS models have dedicated substructures to carry out certain procedures, such as judging the similarity of two patterns, the behavior of these models is largely based on the patterns that are learned by its control memory. This focus on making behavior largely dependent on patterns stored in the network’s memory, rather than on the network’s structure or “hardware,” is a break from previous neural networks and is intended to make GALIS models more generalizable: their behavior can be changed by adjusting which sequences are learned rather than by adjusting the structure of the model itself. This also allows a model’s behaviors to be dynamically modified during task performance (Long et al., 1998) by adding or removing items from the instruction memory rather than changing the network architecture, something that is an important step toward more autonomous intelligent agents.

The second hypothesis is that each region in the cortical network can usefully be viewed as an attractor neural network, i.e., as a dynamical system whose activity is continuously being driven towards certain preferred states. Attractor networks have been used previously in cognitive control models (e.g., Farrell and Lewandowsky, 2002; Hoshino et al., 1997; Jones and Polk, 2002), but usually they are limited to dealing with only fixed-point attractors. However, if working memory is to accommodate procedural information that supports cognitive control, it must also be able to store attractors that are linked together as temporal sequences. In other words, a model region must be capable of switching dynamically from one fixed point attractor state to another. Various techniques have been used to add dynamism like this to attractor nets, including dynamic thresholds, negative feedback, and Hebbian unlearning (Brown et al., 2000; Horn and Usher, 1992; Katori et al., 2011; Tsuda, 2001; Winder et al., 2009). Model cortical regions in GALIS consist of recurrently connected

neural networks that use temporally asymmetric learning of intra-regional connections weights in a fashion that supports storage of temporal sequences of actions (Sylvester et al., 2011, 2010).

The third hypothesis is that each cortical region can not only exchange information with other cortical regions in the form of activity patterns as is done in many other neural networks, but can also gate other regions' functions and interactions. By *gating* here we mean that a cortical region can turn on/off functions in other regions, or open/close the flow of information between other regions, or enable/suppress learning. The claim here is that this is a core aspect of cognitive control. Such gating interactions might be brought about in part by direct connections between regions, such as the poorly understood "backwards" inter-regional connections that are well documented to exist in primate cortical networks (van Essen et al., 1992). However, these gating actions more likely occur indirectly between biological cortical regions, being implemented via a complex network of subcortical nuclei, including those in the basal ganglia and thalamus (Frank et al., 2001; Sherman and Guillery, 2009; van Essen, 2005), and/or via functional mechanisms such as activity synchronization. Synchronization has been postulated as an effective way to gate information flow between cortical areas (Singer, 2011), and this may contribute to top-down attention mechanisms (Womelsdorf and Fries, 2009). While gating has been used in some previous models of working memory control, such past work has generally incorporated explicit neuroanatomical models of hypothesized subcortical nuclei and their interconnectivity to implement gating actions (e.g., Frank et al., 2001; O'Reilly and Frank, 2006). In contrast, in the GALIS framework the details of implementing gating actions via complex subcortical circuitry, synchronized cortical oscillatory activity, or other mechanisms are suppressed.

Instead, the framework assumes such mechanisms exist and implements them as direct gating interactions between model cortical regions and the pathways that inter-connect these regions.

In summary, inspired by the organization of the primate cerebral cortex, the approach adopted in this dissertation to the problem of learned cognitive control is to build a network of regional neural networks, linked together by gated connections. GALIS models incorporate at least two different types of memory systems: those that store task specific state information (task memory), and those that store the actions and procedures necessary for performing the task (control or instruction memory). Both types of memory are implemented as discrete attractor networks and operate according to the same rules. The adaptive gates throughout GALIS networks control how activity flows between regions. In addition, gates are used to control when connection weights are updated. By opening and closing its gates, a GALIS network can determine when to learn and unlearn stimuli. In addition, GALIS also uses distributed rather than local representations and Hebbian learning rather than error back-propagation, both of which are intended to increase the biological plausibility of the GALIS approach.

1.2 GOALS AND AIMS

The central goal of this research is to develop a neurocomputational model of cognitive control which retains some of the capabilities of symbolic systems, in particular the ability to store procedures of behavior. I hypothesize that the ability of neural networks to base their behavior on the contents of their memory in addition to their overall structure will not only make them easier

to build, but will make them more powerful by allowing them to apply their own learning ability to the procedures they are executing.

To this end the following specific objectives guided this research.

1. Design and build a neural model of working memory using sequential attractor networks. Specifically, augment the standard weights used with an additional set of temporally asymmetric weights to allow an attractor network to move from one basin of attraction to another in a particular order. Test this model against human performance on the Running Span task.
2. Using the techniques developed in achieving the first objective, construct a multi-region model of cognitive control with dual memories: one for sequences of external events and one for sequences of instructions for completing the required task. Link these and other regions in the model by gated connections, with the gates controlled by instruction memory within the model itself. Validate this approach using models for both a simple proof-of-concept task and the n -Back task. With respect to the latter, the model should both match human performance and be flexible enough to change which task condition it is performing during execution, without any modification to model structure, parameters or weights.
3. Develop a visual system to extend the model from dealing with purely abstract stimuli to those situated in a spatial environment. Split the visual information about the external world into object and location data (i.e., simulate ventral and dorsal visual pathways) and be able to recombine and process it by binding the two sets of information together.

This system is tested on a common childhood memory game against both human subjects and a comparable symbolic model.

4. Extend the binding of object and location information from the previous objective to handle multiple features, while also developing a system to allow GALIS models to learn from their own experience based on binary reinforcement signals from the environment. This is tested by creating a network that performs the Wisconsin Card Sort Test on the basis of the instructions it is given, testing to see whether it is able to perform at a higher level after having taken the test several times.

1.3 OVERVIEW

The rest of this dissertation is organized as follows. Chapter 2 presents background information about human working memory and neural network models of it, neural networks for sequential processing more broadly, neural models of cognitive control, and techniques used for gating and fast-weights. Chapter 3 presents my model for sequential working memory using temporally asymmetric weights. This technique underpins the overall GALIS model of cognitive control, and the model of working memory provides an introduction to both the types of tasks considered here and the methods used for them. Chapter 4 presents GALIS models for two basic cognitive control tasks, both using asymmetric weight attractor networks to store memories of external stimuli and internal procedures. One of these tasks, the n -Back problem, taken from cognitive psychology, is compared to human results from that field. Chapter 5 discusses a GALIS model capable of binding ‘what’ and ‘where’ information together in order to carry out a memory test which requires

proactive decisions in a visuospatial environment at the same level as human subjects and a symbolic AI comparison model. Chapter 6 goes further by presenting a GALIS model that can bind multiple features with different values in another task from cognitive psychology — the Wisconsin Card Sort Test (WCST) — and can perform this task at an increasing level as it gains experience. Chapter 7 concludes this dissertation with a summary and discussion of the work’s limitations and possible future avenues for addressing them.

Background

This chapter briefly reviews a selection of prior research that is relevant to the work in this dissertation. The chapter begins with a brief introduction to working memory in humans. The following two sections describe some of the neural network approaches which have been used for modeling working memory specifically and cognitive control more broadly. The final two sections of the chapter discuss neural network methods for sequence processing, and the use of gating in neural networks.

2.1 THE NATURE OF HUMAN WORKING MEMORY

Working memory is the ability to hold, monitor and manipulate information needed for tasks in the mind (Durstewitz et al., 2000).¹ Working memory is of

¹ I use “working memory” to refer to the concept in Psychology and Cognitive Science, which stands in contrast to the way the phrase is occasionally used in Artificial Intelligence research for a database-like component which determines which production rules will fire (Charniak et al., 2013). In this usage, the working memory is sometimes of unlimited capacity, e.g., Forgy (1982) and Miranker (1987). See also Baddeley (2000a) and Richardson et al. (1996), which both discuss the debated non-existence of capacity limits in Newell and Simon’s influential 1972 work.

very limited capacity, short duration, and subject to both decay and interference (Cowan et al., 2005); it must strike a balance between the ability to update rapidly and the competing demand to remain fixed in the presence of spurious or distracting information. The term was introduced by Miller et al. (1960) and adopted by Baddeley and Hitch (1974) to differentiate between a unitary short term memory store and the “three-component model” they introduced. This model consists of three parts: the phonological loop, for handling sound and language; the visuospatial sketchpad, for processing objects and locations; and the central executive, serving as a control system (Baddeley, 2003). This tripartite division has been widely adopted, with many offering further refinements and divisions. The problem of how to store items in sequences is of particular interest within the phonological loop. The visuospatial sketchpad has been fractioned into two separate (Courtney et al., 1996; McCarthy et al., 1996; Smith et al., 1995) but overlapping (Awh and Jonides, 2001) subparts, one for objects and the other for spatial coding. The central executive plays numerous potential roles, particularly the focusing, dividing and switching of attention and the coordination between short term and long term memory. Baddeley’s three-component model still underpins most of the research in computational models of working memory (Baddeley, 2012; Lewandowsky and Farrell, 2003).

More recently Baddeley expanded his model by introducing a fourth component, called the “episodic buffer” (Baddeley, 2000a). The episodic buffer mediates between the other components of working memory and long-term memory and serves to bind together information into coherent episodes. The episodic buffer is also hypothesized to be under control of the central executive.

Not all theories of working memory subscribe to Baddeley’s multi-component model. Cowan’s “embedded process model” categorizes information into three

hierarchical levels (Cowan, 1988, 1995). At the base there is all the information in long-term memory, a subset of that is “active” at any point in time, either consciously or not, and a subset of active information is being consciously attended to. Working memory is not a separate function or apparatus, but the set of “cognitive processes that retain information in an unusually accessible state” (Cowan, 1999, p. 62). This is a qualitative model of working memory, rather than a quantitative one from which detailed predictions can be made.

Several other models such as Interacting Cognitive Subsystems (ICS) and Controlled Automatic Processing (CAP2) take the opposite approach: rather than positing a unified group of information and processing like Cowan, they propose a proliferation of independent modules. The ICS model (Barnard, 1985, 1999; Barnard and Teasdale, 1991) is built on a set of specialized subsystems, each suited for a particular class of information such as acoustics or body position. ICS uses no centralized working memory or executive system. Barnard claims that in theory each subsystem could be built using a neural network, but that it would be computationally prohibitive. CAP2 (Schneider, 1999; Schneider and Chein, 2003; Schneider and Detweiler, 1987; Shedden and Schneider, 1990) is a hybrid neural/symbolic model with a multitude of processing modules, though it uses a purely symbolic executive system. CAP2 is discussed further in Section 2.3.

Working memory is distinct from long-term memory in several ways besides its duration. While the capacity of long-term memory is effectively unbounded, working memory has a very limited capacity, often of three to four items depending on the specific situation (Cowan, 2001; Cowan et al., 2005). Additionally, long-term memory is driven by chemical synaptic change, especially long-term potentiation. These synaptic changes occur over much longer time scales, often

requiring repeated exposure to stimuli to form memories. In contrast, working memory is generally viewed as being based on temporary electrical activity patterns in neurons (Barak and Tsodyks, 2014; Dehaene and Changeux, 1989; Gazzaley and Nobre, 2012; Zipser, 1991). Working memory capacity, unlike that of long-term memory, is linked with general fluid intelligence (Conway et al., 2002; Engle et al., 1999). Increased working memory capacity also has implications for increased judgements of probabilities and ability to generate alternative hypotheses (Dougherty and Hunter, 2003a,b).

Another key aspect of working memory is the need to balance rapid updating vs. stable maintenance (Durstewitz et al., 2000; Goldman-Rakic, 1987). (This is sometimes called the “stability-plasticity dilemma” in the neural networks literature.) Long-term memories can be formed and updated very slowly. However working memory must be plastic enough to respond rapidly to new information yet stable enough to persist in the face of distraction. This is a common issue that must be addressed when modeling working memory.

The pattern formed in working memory by an external stimulus can remain active and stable even after the stimulus has been removed. In fact, such patterns can remain stable in the presence of noise, or distracting stimuli. The mechanism underlying this ability is unknown, though the most common candidate is recurrent excitation within cell assemblies (Compte et al., 2000; Zipser et al., 1993). Two other possibilities are “synfire chains,” which are feedforward-connected loops of neurons (Diesmann et al., 1999), and single neurons capable of bistable activity patterns (Guigon et al., 1995). (See Durstewitz et al. (2000) for a review of these theories.)

Perhaps the most prominent method of resolving the simultaneous need for flexibility and stability is to employ “active gating,” sometimes also known

as “adaptive” or “dynamic” gating (Braver and Cohen, 2000; Frank et al., 2001; O’Reilly et al., 1997, 2002). When such gates are open representations in the working memory are free to update and when closed activity patterns are protected from interference. This is captured in part by the LSTM recurrent neural network architecture, in which “memory cells” have nodes which can close off their input or output as well as resetting the cell to “forget” the value being remembered (Hochreiter and Schmidhuber, 1997). This enables a memory cell to store values for an arbitrary length of time but also to change stored values quickly. Furthermore, it is hypothesized that the dopaminergic system in the prefrontal cortex and basal ganglia is capable of modulating behavior in a way very similar to active gating (Cohen and O’Reilly, 1996; Hazy et al., 2007).

The previous section outlines some of the basic characteristics of human working memory. Among these are a limited capacity, a short duration, persistence in the absence of external stimulus or presence of distractions, and the ability to rapidly update state. It is also widely hypothesized that working memory is divided into functional components as well as being tightly integrated with some sort of executive control processes, although the form of the division as well as the nature of the executive are both debated. These characteristics are all elements which should guide the construction of models of working memory.

2.2 PAST NEURAL NETWORK MODELS OF WORKING MEMORY

Attractor networks are often used to model working memory. Attractor networks are recurrent neural networks whose state dynamically shifts until settling into a stable pattern, which may be fixed, cyclical, or chaotic. One such attractor network model is that of Jones and Polk (2002), which uses real-valued attractor networks for serial recall. One positive aspect of this model is the degree to which it gives three important assumptions about neural cognition, namely the use of intra-layer recurrent connections, distributed representations, and Hebbian learning. Jones and Polk’s model is composed of three sections: a “position” network, an “item” network, and a set of “association” networks. The position network stores patterns representing ‘first,’ ‘second,’ and so on, while the item networks trains the actual items being stored. A sequence is stored by training the connections between the position and item networks. It is the learned correlation between the position and item patterns which make recall possible. The association networks exist to strengthen the attractors in the item network through excitatory connections. Each association network is responsible for the memory of a single possible item. For example, a model which was tasked with remembering sequences of letters would have one association network whose only memory is ‘A’, another whose sole memory is ‘B’, and so on. This is the major weakness of this model: an additional layer is needed for every item which might potentially be stored.

Kesner et al. (2000) identifies four uses for attractor networks in the brain. These are: (*i*) short-term and working memory by maintaining activity patterns;

(*ii*) separation or orthogonalization of input patterns; (*iii*) pattern association through correlational learning, and; (*iv*) temporal pattern completion using asymmetric connection strengths. My proposed work makes use of all four of these. Note that temporal pattern completion is a form of pattern recognition more generally, in that assembling a full sequence from partial subsequences is way of restoring full patterns from partial or noisy versions, which attractor networks are widely recognized to do.

It is worth mentioning two other notable models of working memory which use local encoding, especially for the phonological loop and serial recall. One is the “Competitive Queueing Model” (Burgess and Hitch, 1999), which reproduces a wide range of experimental findings about serial recall, but at the cost of high complexity. This was simplified by the “Primacy Model” (Page and Norris, 1998), which accounts for slightly less of the experimental observations (such as grouping effects) but has the benefit of far fewer free parameters. The “Start-End Model” (Henson, 1998) exceeds the explanatory power of the Primacy Model, but requires that the list length be known in advance, which is an unrealistic assumption since humans are more than capable of remembering lists without prior knowledge of the list length, for instance in tasks such as Running Memory Span (Bunting et al., 2006).

OSCAR is a model of working memory for phonological serial recall that does use distributed representations, however its recall is governed by setting a collection of oscillatory timers to the same values they had when the items were first trained, and it is unclear how this would function in a biological system (Brown et al., 2000). Another drawback of OSCAR is shared by TODAM, which is also a neural network model using distributed representations (Lewandowsky and Murdock, 1989; Li and Lewandowsky, 1993). In order to output which item

is being recalled both models must convert back into localist representations in order to combat the noise that their associative memories generate as a result of interference. Having to use modules with local codings of all possible inputs somewhat undermines the use of distributed representations in the first place. Additionally, none of the prior four models display much in the form of a cognitive control system. Instead each is governed exogenously, being commanded when to store items and when to switch to recall mode.

All of these approaches to working memory modeling thus far have had the aim of representing or understanding biological cognitive systems. Pascanu and Jaeger (2011) takes a different approach, building a working memory model for use in a signal processing task. Their model is based on reservoir computing, using an Echo State Network (Jaeger, 2001) augmented with special output nodes they call “WM-units.” These nodes have trainable connections both from the reservoir and between themselves, and fixed recurrent connections back to the reservoir. The WM-units are used to keep track of the number of open curly braces in a stream of handwritten text, while the standard output units identify which letter is currently being presented. Since the distribution of characters in the input stream is a function of the nesting level of the braces, the WM-units provide valuable contextual information for the character recognition. This is an interesting application of the concept of working memory since it fits the commonly used cognitive definition — “the ability to transiently hold and manipulate goal-related information to guide forthcoming action” (Durstewitz et al., 2000) — even though the point of the model is not to do cognitive modeling.

The models described above have several common limitations. The first of these is a lack of internal cognitive control. They rely on some external

force to direct them to add items to memory or to recall them from memory. Secondly, many of these models use local encoding, which is neither biologically realistic, good at generalization, or able to scale well. Finally, even some of the models which use distributed representations suffer from scaling problems since they require additional units, and often whole additional layers, in order to encode more values in memory. Basing GALIS on a network of gated sequential attractor networks overcomes these limitations. GALIS's control module is able to guide the rest of the network's activity in performance of the tasks it has been trained on without relying on exogenous control instructions. Gating is used not just as a method of balancing the stability and plasticity of working memory representations, but also to direct the flow of information and the shape of attractor landscapes. Finally, the attractor network approach frees the model from having to devote additional substrate for each new item added to memory since the attractor network can hold multiple patterns in memory.

2.3 PAST NEURAL MODELS FOR COGNITIVE CONTROL

Compared to artificial neural networks, symbolic AI systems excel at modeling executive behaviors like decision making due to the ease with which they can do things like searching, representing working memory, and variable binding. And yet biological neural systems are completely capable of these sorts of executive functions. This section outlines some of the attempts to build neural networks which are capable of performing tasks requiring cognitive control. In addition to the benefits of bringing the capabilities of the neural paradigm closer in line with the symbolic one, and potentially shedding light on some of

the more mysterious, higher functions of human cognition, exploring neural network models of goal-directed behavior has another advantage. By improving on some of the inherent weaknesses of neural systems one could make better use of some of their strengths, such as their natural facility with generalization or partial pattern matching.

One of the dominant forms of non-neural cognitive models is the “production system,” exemplified by such architectures as ACT-R (Anderson, 1996; Anderson et al., 2004; Anderson and Lebiere, 1998). Production systems take the form of *if-then* rules. Productions are activated or “fired” whenever their preconditions are met. While powerful, the current symbolic implementations of such systems have little basis in biological reality and map poorly onto the known functioning and structure of the brain. Neurally-based architectures have some other advantages over symbolic ones besides their correspondence to biological systems. For instance, neural networks can deal with partial matches with ease. Neural representations are also easier to learn than symbolic ones, since the “discrete and fragile” nature of the latter causes them to be more brittle, making iterative improvements to representations difficult (O’Reilly and Busby, 2002, §2.1).

There have been several efforts to bridge the gap between symbolic and neural systems. CAP2 is a hybrid system, using both neural components, such as associative memories, and symbolic components, such as buffers and production rules (Schneider, 1999; Schneider and Chein, 2003; Schneider and Detweiler, 1987; Shedden and Schneider, 1990). It is difficult to tell how much of each paradigm is used, since the executive system is described as potentially being an Elman-style Simple Recurrent Network but also described elsewhere as being implemented as a symbolic rule system (Schneider and Oliver, 1989,

p. 7). Other control modules, such as the “Episodic Store,” are also symbolic. What neural modules are used are trained with Error Backpropagation. One of the key control modules, the “Goal Processor,” is inherently sequential, like that of GALIS. It influences the operation of other modules by adjusting gain parameters, which can be seen as a primitive form of gating. Details on how the system operates are scant, however.

ACT-RN took a different approach to reconciling neural and symbolic production system by attempting to re-implement ACT-R with neural components while maintaining the same overall organization and operation (Lebiere and Anderson, 1993). ACT-RN required an extra node for every “chunk” of information in its declarative memory as well as an extra node for every production rule it knew. As a result of these and other drawbacks ACT-RN has been declared to be “not of practical use” by its own creators (Jilk et al., 2008, p. 202). One interesting aspect of ACT-RN is the way it handled goals in a real-valued Hopfield memory. When a subgoal is identified the goal memory learns the subgoal’s correlation with the current goal. When the current goal is complete, this correlation can be used to retrieve the parent goal. The correlation between the two is then unlearned, returning the goal memory to roughly the same state it had before the subgoal was added.

ACT-RN was not unique in mimicking the structure of a symbolic production system using neural building blocks rather than crafting the architecture from a neural perspective from the start. One such endeavor was the Distributed Connectionist Production System (DCPS), which was built as a demonstration that a coarsely-coded neural network can implement a restricted set of simple production rules, including a limited amount of variable binding (Touretzky and Hinton, 1988). This is achieved through a set of five modules, four of which

are a modified form of Boltzmann machines. The memory module is composed of simple latches, and these nodes do no computation of their own. The other four—used to isolate particular clauses in the production rules, represent the current rule and bind variables between clauses—all employ competitive dynamics. The connections in and between modules were hard-coded into the network rather than being trained. Additionally the system contained no control elements: all executive functioning was external to the model.

DCPS was later the basis of a connectionist model for symbol processing named “BoltzCONS” (Touretzky, 1990). BoltzCONS is similar to DCPS in structure but it is designed for manipulating Lisp-style data structures and symbol processing rather than production rules. Using competition between nodes BoltzCONS implements associative memory version of linked lists, stacks and trees. Also like DCPS all connections are hardwired in advance and executive function is exogenous.

There have been attempts at building neural models capable of implementing production system-style rules that do not mimic the top-level functioning and organization of symbolic architectures (Kiela, 2011; Kriete et al., 2013; Lamb, 2008; Townsend et al., 2014). One such a system is given by Simen et al. (2010), which uses leaky integrator nodes to represent populations of neurons (see also Simen and Polk, 2009). By setting the weights on the recurrent self-connections of these nodes to $w_{ii} > 1.0$ they behave like bistable switches. The discrete behavior of these switches can then be used to build assemblies which implement a production rule. The input weights to the assembly recognize the antecedent of the rule, and the switch node can then be flipped on or off to signify the consequent. One drawback of this system is that it requires one assembly for every rule in the system. Another problem is that there is no known learning

rule to learn the appropriate connection strengths, so the system must be hard-wired by hand.

In contrast to the attempts to replicate production system architectures with neural building blocks, Dehaene and Changeux (1997) presented an all-neural model, sometimes called DC97, capable of solving Towers of London problems. The network is organized into three hierarchical levels, one each for “gestures,” such as pointing to a location, “operations,” such as moving a particular disk, and “plans,” for higher level abstraction such as determining if a goal condition has been met. The patterns in each layer are kept stable by a combination of competitive dynamics and recurrent excitation. One criticism of DC97 is that it uses a very local conjunctive coding scheme, which is both undesirable from the perspective of biological fidelity and can lead to problems of combinatorial explosion as more entities must be encoded. DC97 also has no learning procedure, making it hard-wired for performing only Towers of London tasks; it is incapable of generalizing to even similar problems such as Towers of Hanoi.

Polk et al. (2002) also criticized DC97 for being “method-specific” because it implements only greedy depth-first searches of the state space, albeit with some look-ahead capability. In that paper, Polk et al. present their own model of the Towers of London task, which is built around Hopfield networks. Each variable used in the model gets its own Hopfield net, and each network is trained using Hebbian learning so that all possible values of the corresponding variable are attractor states of the network. Production rules are then implemented by training the connections between the antecedent and consequent variables’ networks. For instance, a rule `if color=red then size=medium` would involve connections from the `color` network to the `size` network which biases the latter

towards the `medium` state whenever the former is near the attractor `red`. This system can accommodate goals by introducing extra biases on the consequent networks which further deforms the attractor landscape, enlarging the attractor basins corresponding to the desired outcomes. The major drawback with this system is that it requires a layer of nodes for every variable. This both scales poorly with task complexity and locks the model in to representing just the task it was designed for.

Another alternative to the depth-first search of DC97 goal-seeking is the approach of Schmajuk and Thieme (1992). It was built for maze navigation, but can be applied to any problem describable by a directed state graph. Their model is divided in two components, the “cognitive system,” which builds a topological map of the space by linking adjacent locations, and the “action system,” to handle action selection. The cognitive map builds heteroassociative correlations between “places” and “views,” which are the locations immediately adjacent to the current place. These associations are then used to make predictions about what the results of taking certain paths will be. It is possible to make long-range predictions by feeding the output view back in to the map as a place and repeating the process.

Schmajuk and Thieme’s model has a particularly interesting property: it exhibits “latent learning,” which is the ability to learn about the environment in the absence of any reinforcing rewards (Tolman and Honzik, 1930). If it is allowed to wander through a maze without any reward being present at the goal it still builds a cognitive map of the environment. Then when it is re-introduced to the maze with a reward present it is able to act on the knowledge previously acquired. Few models are able to successfully learn both with and without reward signals.

Many control architectures fall prey to a homuncular fallacy: their proposed solutions to the problem of cognitive control themselves require an external controller (van Veen and Carter, 2006). If the controller itself needs to be controlled, as if driven by a homunculus, then as many questions are raised as answered: what is controlling the homunculus? Addressing this issue was the motivation behind the PBWM model (O’Reilly and Frank, 2006). It combines modules inspired by the prefrontal cortex and subcortical structures, especially the basal ganglia, using actively gated connections to implement an actor-critic architecture for reinforcement learning. This method, called the PVLV algorithm (“primary value, learned value”), is inspired by Pavlovian learning in psychology and is an alternative to TD-learning (O’Reilly et al., 2007, 2014; *but see* Houk, 2007). It is used to enable the basal ganglia module to learn when to gate connections.

Though this learning system goes a long way to sidestepping the problems of homuncular control by allowing the control dynamics to emerge, the architecture used for these models is very hard-wired to the particular task at hand and relies on conjunctive, local representations. For instance, in their model of the Wisconsin Card Sort Test (WCST) there is a single prefrontal cortex (PFC) node each for the shape and line dimensions, and the ventral tegmental area was represented by another solitary node (O’Reilly et al., 2002; Rougier et al., 2005). O’Reilly and colleagues have made occasional efforts to address this reliance on local representation, such as O’Reilly and Busby (2002). However, three of the six modules in that model still used local representation. In the modules that did use distributed representations, objects were encoded by activating just two of eight nodes, rather than the one of eight that would be used for local encoding.

GALIS differs from PBWM in several ways. First of all, PBWM uses firing rate neurons, while GALIS uses a more abstract paradigm in which each node represents the combined, often binary activity of a group of neurons. Secondly, GALIS makes a stronger commitment to distributed representations. PBWM and GALIS also differ in their approach to gating. PBWM’s gates are based on the binary state of an intra-node latch, while gates in GALIS modulate the input entering a node using continuous values. PBWM uses gating for shielding the contents of working memory from update, while GALIS uses gating for controlling flow of information between modules more generally, including inputs, outputs, and biasing attractor networks, in addition to updating working memory. Finally, it is my goal for GALIS to require fewer task-specific changes to its architecture than PBWM does.

The PBWM model of Chatham et al. (2011) is a particular contrast to the work I present in Section 4.3. The authors present a model of a sequential memory task called *n*-Back, but the inputs to their model consist of both the letter to be remembered *and* that letter’s serial order.² So while the GALIS model described in this dissertation receives a stream of stimuli such as *A S D F S G . . .* no matter what the value of *n* is, Chatham et al.’s PBWM model receives *A1 S2 D1 F2 S1 G2 . . .* if it is supposed to perform 2-back but *A1 S2 D3 F1 S2 G3 . . .* if the task is 3-back. Structuring the inputs this way removes some of the burden from the model to determine which prior stimuli the current one should be compared to. In addition, it uses an iterative approach to training, while GALIS use one-shot Hebbian learning. Further, the PBWM model prevents interference between memories by learning to explicitly over-write old memories with the new ones which occur at the same position in

² *q.v.* § 4.3 for an explanation of *n*-Back.

the period (so that, for instance, G3 displaces D3 in the example above), while GALIS uses weight decay and Hebbian unlearning to minimize interference from older, irrelevant stimuli.

Kaplan et al. (2006) developed a biologically-motivated model of cognitive control for the WCST which used a Hopfield network as a working memory and gating to control maintenance and updating. Though it was capable of modifying behavior in response to feedback from the environment, it is limited by its extremely small size and local encoding: the entire working memory was composed of only four nodes.

Morton and Munakata (2001) present a neural network model of cognitive control which adopts an “active-latent account,” dividing control mechanisms and memory into two types. Active representations in the PFC can overcome habituated, latent ones in the posterior cortex. Morton and Munakata link active representations to activity patterns which can be maintained even in the absence of their antecedent stimulus, and latent representations to weight changes. Flexibility of control is related to the relative strengths of active and latent representations. Their model generalizes well, being able to model both the WCST and a verbal interpretation task with minimal architectural changes, but representations are extremely localized.

There is neuroanatomical evidence of “loops” running from the PFC to the basal ganglia to the thalamus and back to the PFC. These loops have been the subject of various models, for instance Amos (2000) assigns the PFC module to maintain information including the most recent stimulus and the currently operative rule, and the basal ganglia modules to integration of PFC outputs and gating possible actions. (The thalamus serves as an output in this model,

as it does not project back to the cortical layer.) It also uses local coding, with one neuron per feature-value pair in the WCST.

Another loop-based model is Monchi and Taylor (1999), which presents a spiking neuron model designed specifically to align with fMRI data (Monchi et al., 2001). While the architecture used can be fairly easily adapted to both Delayed Response Tasks (Petrides, 1994) and the WCST, which shows some versatility, it is both locally coded and hard-wired to the task selected, with all weights and connections set by hand, without learning algorithms. This model can also be artificially “lesioned” in order to study such neurological abnormalities as schizophrenia and Parkinson’s disease (Monchi, 2000).

Dehaene et al. (1998) created a neural model of cognitive control for the Stroop task based around a “global workspace” for effortful mental operations. This is a limited implementation of Baars’ global workspace theory (Baars, 1983, 2002), which revolves around consciousness and awareness. Dehaene et al. postulate two different types of computational “spaces”: specialized ones for roles like perception and motor control, and a general space to mobilize or suppress the specialized modules. This formation also relies heavily on a gating concept, with only some nodes of some modules being granted access to the global workspace. Using semi-supervised learning, this model can learn to perform the Stroop task without any special rule coding units or pre-programmed behaviors. However, once again local encoding were used and the architecture is specific to Stroop. Furthermore, once the network has learned to perform Stroop, it could not learn another task.

The problem of action selection recurs in all executive systems. Many neural network approaches are based on Gurney et al. (2001a,b), which deals with a model of action selection based on the basal ganglia (e.g., Beiser and Houk,

1998; Frank, 2005; Stewart and Eliasmith, 2011). A signal is selected through an off-center, on-surround pattern of activation caused by the combination of diffuse excitatory connections with concentrated, topographic inhibitory connections. (Off-center, on-surround is the opposite arrangement of that used for typically used for selecting winners, such as in self-organizing maps. However the basal ganglia have inhibitory output so it is the *least* active signal which designates the action to be selected as low basal ganglia output will lead to disinhibition of the corresponding action.) Interestingly, the selected action is output twice: once to the PFC via the thalamus, so that it may be acted upon, and once back to the basal ganglia. It is hypothesized that this second output is used to adjust the action selection process itself, but the exact mechanism for this is unclear.

In order to select from a set of actions it is necessary to have already learned something about them and their expected outcomes. Fortunately, the basal ganglia has also been linked to a reinforcement learning process (Barto, 1995; Cohen, 2008; Rivest et al., 2004). There is a particularly close alignment possible between the actor-critic model of TD-learning (Sutton, 1988) and the functioning of the basal ganglia.³ By adding in circuits analogous to cortical regions, Botvinick et al. (2009) recently built a model capable of “hierarchical reinforcement learning,” which allows the network to learn reusable subroutines called “options.” This temporal abstraction is very helpful for alleviating the temporal credit assignment problem.

The actor-critic framework is not the only version of TD-learning linked to the basal ganglia. Walsh and Anderson (2010) examine three differed TD-

³ There is an impressive degree of correspondence between the actor and the dorsolateral striatum and between the critic and the ventral striatum and dopaminergic system (Doya, 1999).

learning models, actor-critic, Q-learning and SARSA, all of which are supported by different neural evidence. (Respectively, neuroimaging studies (O’Doherty et al., 2004), dopamine levels in rats (Roesch et al., 2007), and single cell readings in monkeys (Morris et al., 2006).) It was Walsh and Anderson’s opinion that the Q-learning model accounted best for both neural and behavioral data.

Finally, it is worth noting the work of Eliasmith and colleagues (Eliasmith, 2013; Eliasmith et al., 2012). They have built models of numerous tasks using spiking neuron implementations of Holographic Reduced Representations (HRR; Plate, 1995, 2003b), which is a type of Vector Symbolic Architecture (VSA; Gayler, 2003). VSAs represent symbols as high dimensional vectors, such as points on a unit hypersphere. Individual vectors can be combined in two different ways, superposition (+) and binding (\otimes). The superposition of two vectors is accomplished through vector addition and gives a result which is similar to both operands. The binding operation is done with circular convolution and gives a result which is dissimilar to both operands. In addition, there is an approximate inverse function ($*$) which allows vectors to be unbound. All of these operations can be computed with spiking neural networks, but there is no learning rule to do so. Rather, the appropriate weight matrices are determined analytically in advance and hard-coded into the system (Eliasmith and Anderson, 2003). Crucially, all of these operation produce outputs which are the same dimensionality as their inputs. This allows operations to be combined to produce structured representations, such as attribute-value bindings. For instance, a binding of X to attribute A_1 and Y to A_2 could be represented by a vector $R = A_1 \otimes X + A_2 \otimes Y$. To get the value of A_1 , simply bind R with the inverse A_1^* .

This general system has been used for the Wason card task (Eliasmith, 2005), sequence memory (Choo and Eliasmith, 2010), production rules (Stewart et al., 2010), Raven’s Matrices (Rasmussen and Eliasmith, 2010), and Towers of Hanoi (Stewart and Eliasmith, 2011). While this is a very impressive range of complex problems this approach is capable of addressing, there are some weaknesses as well. One of these has been mentioned already: there is no learning rule to produce the transformations necessary for the binding, superposition and inverse operations. Another is related to recovering values from attribute-value bindings. Binding with the inverse of the attribute yields only an approximation of the value being sought. Recovering the actual value requires a “cleanup memory” (Stewart et al., 2011) to remove excess noise. In order to do this the cleanup memory must be pre-trained with all the atomic symbols that could potentially be recognized, limiting the model to a distinct vocabulary of symbols. Furthermore, there needs to be a separate set of nodes dedicated to recognizing each symbol in memory. There are other aspects of several of these models which requires the size of the model to scale up in proportion to behavioral complexity. For instance, each rule necessary to perform the Towers of Hanoi task needs its own assembly of nodes. Finally, these models entail a great deal of computational complexity. The Towers of Hanoi model, for instance, uses 150,640 leaky-integrate-and-fire spiking nodes, each of which processes a 128-dimensional vector.

Unlike the models discussed at the beginning of this section, namely ACT-RN, DCPS and BoltzCONS, GALIS is not attempting to recreate a symbolic architecture with neural components. Neither is it a hybrid model, using both neural and symbolic systems when convenient, like CAP2. Rather, GALIS is an attempt to model executive function from the ground-up, drawing on

neurobiology rather than production systems for inspiration. Many similar neural models of cognitive control, such as DC97, the work of Simen and Polk, and that of Eliasmith and colleagues, suffer from a common problem related to scaling. Each of these systems requires an additional set of nodes for every stimulus (e.g., Jones and Polk, 2002; Stewart et al., 2011) or rule (e.g., Simen et al., 2010; Stewart and Eliasmith, 2011) the model must respond to. This leads to several issues. The first is a computational problem, because such models scale poorly to more complex tasks and environments. The second is the conflict that arises with what we know of biological neural networks, which make heavy re-use of substrate (Anderson, 2010). The third is the brittleness this introduces to the model. The designer must know in advance how many symbols will be in the input set, or how many actions will be required to perform the task. GALIS avoids these issues by not requiring such linear increases in the neural substrate with every additional behavior.

There is another common limitation of biologically-inspired models of cognitive control, such as those O'Reilly and colleagues as well as the work of Dehaene et al. (1998), which is that they are very specific to the task being modeled. This hard-wired nature is often combined with, or sometimes a consequence of, the local, conjunctive encoding schemes used. GALIS relies on distributed representations and a more general architecture which should minimize the number of changes needed to model different tasks. This also allows other advantages of distributed representations including ease of learning and noise resistance.

GALIS does not yet incorporate features of the many models which use analogs of the basal ganglia or thalamocortical loops for action selection or reinforcement learning. However, this could be added in future expansions.

None of the models which do use these subcortical approaches are coupled with attractor networks like GALIS is. Other authors take the basal ganglia and thalamic nuclei to be the locus of gating activity, but gating can also be seen to be a result of interactions between these subcortical elements and the cortex. At least for the time being I am abrogating these biological details and treating gating as if it were a cortical function (indirectly, via a subcortical mechanism), making the basal ganglia and thalamus implicit in GALIS.

Finally, I believe GALIS is unique in using a memory not just to store a record of past inputs, but also storing the instructions needed to execute a task. Many models have weight matrices which encode the mappings necessary for a task, but as far as I know none of them encode the task instructions in the states of nodes. By processing both memories of inputs and memories of procedures using the same techniques GALIS adopts a form of code-data equivalence.

As a consequence of this explicit task memory, GALIS learns to perform tasks rather than having behaviors hard-coded into the model itself. Because different behaviors result from different instruction memories, which in turn result from different training data, there is much more possibility for generality and flexibility. For instance, GALIS should be useful in studying transfer effects between tasks, since the difference between two models of different tasks will lie primarily in the contents of their instruction memories and less in their architecture, making them much more comparable.

2.4 PAST NEURAL NETWORKS FOR SEQUENCE PROCESSING

This section is limited to just those studies which are relevant to either the subject of my work (executive functions of cognition) or the methods I employ (principally temporally asymmetric learning). For a more comprehensive review, see Kremer (2001).

Within the psychological domain neural networks have become a major way to model serial recall (Brown et al., 2000; Burgess et al., 1999; Page and Norris, 1998; Pascanu and Jaeger, 2011; Ponzi, 2008; Verduzco-Flores et al., 2012). However, these models do not agree on a unified theoretical perspective. The most common approaches can be divided into inter-item, ordinal and positional theories (Henson and Burgess, 1997). The latter two approaches are sometimes called “context-based” accounts. Inter-item models associate items in the sequence with each other and manage recall by “connecting the dots” between consecutive items. For this reason the most common of these theories are called “chaining” models. Ordinal models such as Page and Norris’ (1998) Primacy Model store sequences along a single dimension such as the overall strength of their representation, so that the first item is the most active, the second item is the next most active, and so on. Positional models associate items with some context information such as a unique pattern indicating its ordinal position (Anderson and Matessa, 1997), its relative distance from the start or end of the list (Henson, 1998; Houghton, 1990), or the state of an oscillatory neural timer at the moment the item was presented (Brown et al.,

2000; Burgess et al., 1999). These three subtypes of positional models are called absolute, relative and temporal, respectively.

Inter-item accounts, typically based on recurrent neural networks, have better support from the neuroscience data, but context-based accounts have better support from the behavioral data. The recurrent neural network model of Botvinick and Plaut (2006) is an attempt to reconcile this by creating an inter-item system that matches some important behavioral observations which earlier chaining accounts could not. While it does meet this goal, it does so using Backpropagation Through Time with teacher forcing, which is not very biologically plausible. Botvinick and Plaut’s method is to train a modified Simple Recurrent Network to store and echo back any sequences to which it is exposed. The stimuli presented during trials are represented only by the network’s activity and not by its connection weights. Rather than the network’s weights learning the particular sequence as it is presented, they learn the behavior “store this sequence and then repeat it as output,” in advance over the course of tens of thousands of training samples. Note that local encodings are used for the input and output, though not for the internal, hidden representations. The authors claim that the local encoding is not necessary, but it is unclear to what degree the model relies on it. If distributed representations were used it is possible that a prohibitive amount of training would be required because of the concomitant increase in the number of potential patterns that the network would have to learn to process (Plate, 2003a).

As mentioned previously, attractor networks are commonly used to model working memory (e.g., Maniadas et al., 2012). While this approach is effective and has generated substantial theoretical and experimental analysis, it is

typically limited to maintaining only a single pattern at a time in short-term memory. This restriction makes sequence processing difficult. In response to these and other concerns, a number of oscillatory memory models have been created and studied during the last several years. In these models, items in memory are typically represented as rhythmic network activity in which multiple memory patterns are simultaneously present in the same neural substrate. This is possible because the networks activity oscillates between activity states representing different stored patterns.

A diverse set of oscillatory memory models exists. Some are based on theories about the mechanisms underlying theta/gamma activity in specific brain regions such as the hippocampus or neocortex (Hasselmo et al., 2002; Ingber, 1995; Koene and Hasselmo, 2007; Lisman and Idiart, 1995), others use individual spiking neurons (Raffone and Wolters, 2001), while still others have adopted more abstract approach such as Wilson-Cowan oscillators (Chakravarthy and Ghosh, 1996; Hayashi, 1994; Wang, 1995).

A particularly simple and elegant approach to creating oscillatory sequence-processing memories is based on minimally modifying Hebbian associative memories having fixed-point attractor states so that they become oscillatory. For example, Horn and Usher (1991, 1992) produced a simple oscillatory memory by introducing “dynamic thresholds” into Hopfield networks (Amit, 1989; Hopfield, 1982). With this approach, whenever a node has a particular activity level ± 1 , the threshold of that node gradually changes so that eventually the node switches its activity level to the complementary value. When such a network is presented with an input that is a superposition of multiple stored memories, it oscillates between activity states that represent these individual memories, thereby indicating its recall of the memories in parallel. Similar

behaviors have been produced based upon Hopfield networks modified to use dynamic synapses (Pantic et al., 2002) or negative feedback with asymmetric connection weights (Brown et al., 2000).

Horn and Usher’s approach was extended by Winder et al. (2009) and Reggia et al. (2009) to include rapid decay of connection weights. This weight decay allows the network’s activation to be influenced by the order in which stimuli are presented, something that is not the case with classical Hopfield networks. While the combination of dynamic thresholds and decay enabled networks to match the position-specific recall rates of human subjects, the order in which the stimuli were recalled by the model was arbitrary.

The Serial-Order-in-a-Box (SOB) model is an alternate approach to storing sequences in attractor networks (Farrell and Lewandowsky, 2002). Items are stored using decreasing learning rates, so that the earlier an item appears in the sequences the larger its basin of attraction will be. When the network is put in a random state, it is likely to be drawn to the first, largest basin. Rather than using dynamic thresholds to induce a transition to another attractor, the networks weights are adjusted after each item is recalled to suppress the most recently recollected pattern. This adjustment of weights after each item is recalled has several drawbacks, including making rehearsal difficult. Additionally, SOB requires that items be represented by orthogonal patterns.

One technique that has been used to capture sequential patterns is temporally asymmetric weights (Abbott and Blum, 1991; Blum and Abbott, 1996; Rao and Sejnowski, 2001). This is a Hebbian learning process which bases connection strengths not on the correlation between concurrent activity in pairs of nodes, but on the *consecutive* activity of those nodes. That is, learning is based on a the correlation between the current activity and other nodes’

activity during the prior time step. Often these models deal with the timing of spike trains in spiking neuron models (e.g., Abbott and Song, 1999; Gerstner et al., 1993). Rao and Sejnowski (2001) and Dayan (2002) also link asymmetric weights to temporal difference learning. These studies have support in recent neural experimental evidence (Bi and Poo, 1998, 2001; Markram et al., 1997; Zhang et al., 1998).

Schulz and Reggia (2004) took a very different approach, modifying a self-organizing map (SOM) to represent sequential information using temporally asymmetric weights. The input to each node in the map is a function of the current stimulus as well as the activity state which resulted from the previous inputs. Weight updates are defined in such a way as to correlate these two components of input using one-shot Hebbian learning. This is one of several modifications made to SOMs to enable sequence processing, and many of the other approaches are reviewed in that paper.

Hoshino et al. (1997) also adopt an approach using temporally asymmetric Hebbian learning, but do not use it to form a separate weight matrix. Rather, weights are generated using the sum of both standard and asymmetric Hebbian learning components. This technique is used to form networks whose dynamics take the form of “itinerant attractors,” in which the network’s state moves in state space between different attractor basins, each one of which represents one stored pattern (Tsuda, 2001). By adjusting the relative contributions of the symmetric and asymmetric learning, the network will either move between attractors in the order they were trained or in a random order, like SOB. One drawback of this approach is that the weight matrix must be modified in order to induce a transition from one basin to another. This is done by reducing

the size of the current attractor basin using anti-Hebbian learning so that the network transitions into a different one.

Dealing with sequence processing is central to GALIS, since neither perceptions nor actions occur in isolation, but rather are inextricably embedded in the sequence of perceptions and actions which occur around them (Elman, 1990). Just as some AI researchers focus on agents situated in physical environments, I believe cognition must be situated in a temporal environment. I approach this serial nature of cognition by integrating methods used in many of the models discussed in this section. I adopt an inter-item approach, exemplified by psychological research such as that of Botvinick and Plaut's, in contrast to the context-based approaches of many qualitative models. However, I expand on Botvinick and Plaut's work by using distributed representations and correlational learning, which are both more biologically plausible, less fragile, and better able to generalize than the local representations they employ. I also draw on attractor network models like SOB, as well as the past work in computer science and neuroscience in using temporally asymmetric learning to process sequences. By combining these methods with attractor networks I can harness their ability to generalize and restore full patterns from incomplete versions and use this to build robust representations of sequences.

2.5 FAST WEIGHTS & GATING

Error backpropagation is a very successful type of artificial neural network learning technique, shown to achieve both high accuracy and good generalization capability on many tasks. However, there is very little evidence for error propagation signals in biological neural networks. Additionally, back-

propagation is by nature a slow training method, requiring highly repeated iterative presentation of each training stimulus. As such, it makes a poor model for biological short term working memory, which does not require long-term, repeated exposure of a stimulus in order to insert it into memory. Indeed, working memory is defined in part by its ability to be rapidly updated. It is important, therefore, to examine other neural network models which allow for near instant training and fast updating of weights.

The most commonly known technique in this category is “one shot” Hebbian learning. This is a learning technique often used in Hopfield networks to form associative memories (Hopfield, 1982). Only a single presentation is required to store a stimulus in memory. The downside to this rapid storage is that stored stimuli can interfere with previous memories, especially if multiple stimuli are sufficiently similar. As a result Hopfield networks have a small but variable memory capacity, with capacity being a function of the particular items being stored and the network size. This restriction is not unlike biological working memory however.

One of the earliest fast-weight paradigms is presented by Hinton and Plaut (1987). Their system had two weights on every connection, both trained by error backpropagation. One would have a much higher learning rate than the other, however, resulting in much faster adjustments. The effective weight at any point in time would simply be the sum of the fast and slow weights. Hinton and Plaut outline several applications, but concentrate on presenting a method for reducing the interference that training a network on new patterns has on the patterns already stored. This is an interesting concept, but the “fast weights” are really only fast in comparison to the slow ones from this model, as they still trained over hundreds of epochs.

Kak has developed a neural network paradigm requiring only a single training step for each input called the “Corner Classification” method (Kak, 1993, 1998, 1999).⁴ The training method for CC networks requires only a single pass through the training data with minimal computation at each step to determine connection strengths. The major drawback is that a distinct hidden node is required for each training sample. Such local representation is reminiscent of “grandmother cells,” and a large number of nodes would be required for a complicated problem domain. (Note that training is very quick in relation to the number of nodes, so large networks are more feasible than they would be using a different training method.) CC networks can generalize but only within a fixed-width hypercube around each training sample. Additionally the radius of generalization is constant for the entire network. Finally, CC networks operate only on binary data, and are best at two class discrimination problems, though tasks with more classes can be handled by building several networks in parallel, each of which is trained to recognize a single class.

More recently Kak extended CC networks to a model called “Fast Classifier Networks” which allow for real-valued input and different degrees of generalization in different subregions of the input space (Kak, 2002; Tang and Kak, 2002). FC nets retain the ability to train quickly, though they need two passes through the training data rather than one. The first is used to set weights in the network and the second is used to determine what the radius of generalization will be for each hidden node. The requirement for a hidden node per training sample is retained. The other major change is the addition of a “Rule Base,”

⁴ This is not to be confused with the corner classification *problem*, a data set used to test neural networks. Roughly speaking, Kak’s corner classification method converts every problem into an instance of the corner classification problem, and so the problem and the method share a name.

making FC nets a hybrid connectionist-symbolic model. The function of the Rule Base, which contains only two rules, is to determine whether the network will act according a single nearest neighbor or a k -nearest neighbor model when evaluating each test sample. The former occurs when the test stimulus is within the generalization region of a hidden node, otherwise the latter is used. As such FC networks are a connectionist implementation of the statistical nearest neighbor technique, and gain their fast training time in a similar way to the “lazy” method of nearest neighbor techniques.

The Restricted Boltzmann Machine is similar to a stochastic version of Hopfield nets in that they are associative, energy minimizing networks. The RBM has also been interpreted as a Product of Experts model, with each hidden node in the RBM being equivalent to one expert (Hinton, 2002). Tieleman and Hinton (2009) proposed a training technique for RBMs which incorporated the RBM’s usual, slowly changing weights with a supplementary set of fast weights which learn and decay more rapidly. Only the standard weights are used to define the energy landscape the network is acting within, while the fast weights are used to define a temporary “overlay” on that landscape. The Markov Chain Monte Carlo method which is used to sample the network’s state for training purposes then operates on this temporary overlay. The energy landscape defined by the normal, slow weights is updated gradually and with decreasing velocity, allowing the network to settle into an attractor basin. The landscape of the fast weights is more dynamic, allowing the MCMC sampling to converge closer to the probability distribution of the network quickly.

“Competitive activation dynamics” is an alternative to lateral inhibitory connections for inducing competition between nodes in connectionist models (Cho and Reggia, 1993; Reggia et al., 1992, 1988). Rather than using internode

connections which explicitly act to create inhibition, this paradigm allows nodes to compete with each other for the activation of upstream nodes. Thus, a node's input is not just dependent on its inputs and associated connection strengths, but also on its own current activity. This creates a "rich-get-richer" dynamic in which activation is allocated to one or a few winning nodes and drained from others. This can be accomplished by having two sets of connection strengths: a set of fixed "resting weights" and another set of "fast weights" which are redefined every time step (Reggia and Edwards, 1990).

Schmidhuber (1992) presented a technique for fast weights involving two parallel feedforward networks. One network — called the "fast" network — is trained to associate the desired inputs and outputs, as usual, while the second network — the "slow" network — is trained to adjust the weights of the fast network. The slow network's outputs are the weight changes the fast network requires to properly map inputs to outputs. This allows extremely rapid updating of the fast network's weights, allowing for high levels of plasticity and dynamic response, but without upsetting the stability of the overall system, because the slow network's weights are still comparatively static and stable. This slow network/fast network arrangement makes possible a form of temporary variable binding, which is a problem artificial neural networks often struggle with. The weights within the fast network bind the inputs, acting as variable addresses or slots, to the outputs, acting as the potential values of the variables. The slow network acts as a controller to rapidly update which fast network outputs (values) are responsive to which fast network inputs (addresses), thereby changing the value of the stored variable. This technique was also used to evolve controllers for a difficult pole-balancing task (Gomez and Schmidhuber, 2005).

Schmidhuber’s fast weight networks can be seen as a precursor to the Long Short-Term Memory (Bakker, 2002; Gers and Schmidhuber, 2000; Hochreiter and Schmidhuber, 1997). The Long Short-Term Memory (LSTM) is a recurrent network for sequential memory which relies on gating. It combines fast training and efficient learning using a specific architecture. The key feature of this architecture is a hidden layer composed of “memory blocks.” Each block has three gates: one controls input to the block, another controls output, and another wipes the state of the block (Gers et al., 2000). These gates allow units’ effects to be changed very rapidly in response to inputs. LSTM training possesses two important properties: locality in both space and time, meaning that weight updates are not dependent on global information about the network nor on information from arbitrarily far in the past. However, this LSTM learning procedure can only be used on the very specific architectures, in particular LSTM networks can not have multiple hidden layers in parallel. Recently, a generalized version of LSTM, called LSTM-g, has been developed which allows more flexibility in architecture, as well as architectures which vary during trials (Monner and Reggia, 2013, 2012). One of the key advances which makes this possible is a shift in how gating is used. Rather than gates being applied to the states of nodes within memory cells, they modulate the connections between them. As mentioned in Section 2.1, LSTM was part of the inspiration for the PBWM model of working memory and cognitive control. Gating is the key feature of PBWM which enables the switch between active maintenance and rapid updating of activity patterns (O’Reilly and Frank, 2006; O’Reilly et al., 2002). Several papers by O’Reilly and colleagues identify gating as a major area of cognitive control research (O’Reilly, 2006; O’Reilly et al., 2010).

The Mixture of Experts (Jacobs et al., 1991b) and Hierarchical Mixture of Experts (Jordan and Jacobs, 1994) methods take the gating approach to fast weight changes further by gating entire networks rather than individual connections. The Mixture of Experts architecture is composed of multiple “expert networks” in addition to a “gating network.” The expert networks are trained like standard feedforward neural networks (often multilayer perceptrons using error backpropagation). The gating network however is trained to learn which of the expert networks is likely to be most accurate for a given input vector. The outputs of the gating network are then used to combine the outputs of the expert networks, weighting the final Mixture of Experts output towards the expert network judged most likely to be correct for the current input. The Hierarchical Mixture of Experts architecture is similar to Mixture of Experts but is arranged in a tree structure. Multiple gating networks are used to select the expert network in the tree most likely to be successful for the current input.

Though weights in the composite network may not be changed rapidly, the behavior of the network can be made to change rapidly based on the output of gating networks. If one gating network outputs a zero signal to the gate of a particular subnetwork then the weights of that subnetwork may as well all be zero, behaviorally. Similarly a high gating signal serves to magnify the output of the subnetwork to which it corresponds.

These frameworks have been used for many different machine learning problems such as document classification (Ruiz and Srinivasan, 2002) and control of industrial plants (Ronco et al., 1998; Ronco and Gawthrop, 1997). In the latter, gated modular networks such as Mixture of Expert systems were found to be especially useful when the environment requires abrupt changes by the controller. In this respect, controlling an industrial plant and controlling a

cognitive system are more similar than they may at first seem, in that they both require a balance between stability and rapid response to external changes.

One other application that is of particular interest for my research is the use of a Mixture of Experts architecture to perform a “what/where” task (Jacobs et al., 1991a). For each input the network is asked which of nine three-by-three patterns is present on a five-by-five grid as well as which of nine possible locations the pattern is centered on. One network successfully learned to specialize on each subtask. This is a good demonstration of the ability of gating to combat interference and distinguish between object and location information. However, I believe the simplicity of the networks involved — the gating “networks” in one case were no more than output nodes with biases — leaves much to be expanded upon.

Unlike many other models of cognitive control, GALIS uses exclusively one-shot learning. This is not only biologically plausible in many situations, but also side-steps one of the major computational costs of many neural networks that use backpropagation techniques, which can take hundreds or thousands of iterations. In addition to connections trained with one-shot learning, which have an inherent “fastness” to them, GALIS incorporates three other aspects of fast weights. The first is present in GALIS in the form of its gated connections. These gates move beyond those in or PBWM by taking continuous values, including negative ones, instead of being in binary open or closed states. The second aspect is fast-weights of the form presented in Schmidhuber (1992). This underlies my approach to parallel visual pathways in Chapter 5. The third is learning methods added to handle set shifting and instruction refinement in Chapter 6, which are more similar to the type discussed in Tieleman and Hinton (2009) Numerous neural networks have made use of one of these fast

weight techniques, especially to solve problems related to the trade-off between stability and plasticity, but none has explored the use of multiple fast weight techniques concurrently.

Associative Memories Based on Temporally Asymmetric Weights

3.1 INTRODUCTION

Recurrent connections combined with the appropriate dynamics enable oscillatory neural networks to produce rhythmic activity patterns. Such oscillatory activity could potentially represent multiple stored patterns simultaneously, rather than the single pattern of a typical fixed-point attractor neural network, and without requiring the addition of hidden “delay” layers used by many recurrent neural networks for sequence processing. However, retrieving these stored patterns in the same order as they were observed—or any other desired order—has proven challenging. The goal of this chapter is to address this challenge through the use of temporally asymmetric weights in attractor networks to create an auto-associative memory that can learn sequences of patterns. This is done in the context of modeling human working memory, one of the fundamental cognitive control capabilities. A model is built which will form the basis of all the memory systems in the remainder of this dissertation. It is capable of matching the recall performance of human subjects on a standard

cognitive psychology memory task (Running Memory Span), it reproduces the recency effect humans exhibit in working memory, and it displays similar position-specific recall rates.

There has been increasing interest in recent years in the development of oscillatory neural network models for a variety of tasks. In contrast to fixed-point attractor networks, which are typically limited to activating a single pattern in memory at a time, oscillating networks have dynamics characterized by recurrent connections leading to persistent rhythmic activity. This allows multiple patterns to be held in the same short-term memory concurrently as the model's state persistently switches between them.

A large variety of oscillating neural models exist. For example, some are based on underlying theta/gamma activity in the hippocampus or neocortex (Hasselmo et al., 2002; Ingber, 1995; Koene and Hasselmo, 2007; Lisman and Idiart, 1995), while others use individual spiking neurons (Raffone and Wolters, 2001). Other more abstract approaches have also been used, for example Wilson-Cowan oscillators (Chakravarthy and Ghosh, 1996; Hayashi, 1994; Wang, 1995). For further examples, see Section 2.4.

The focus here is on modeling short-term working memory, which is active over periods of time on the order of several seconds. A key characteristic of working memory is that it has a very limited capacity, unlike long-term memory (Baddeley, 2000b). Recent studies suggest that this capacity is capped at around four items (Cowan, 2001; Cowan et al., 2005). More specifically the concentration is on working memory for sequential tasks, or those for which the serial order of stimuli is important.

An elegant and parsimonious approach to oscillating working memory models is based on simple modification of Hebbian associative memories with

fixed-point attractors to make them oscillatory. For example, Horn and Usher (1991) developed a basic oscillatory memory by adding “dynamic thresholds” into Hopfield networks. With this approach, the thresholds used to determine the next activity state of a node are continuously changing such that it becomes increasingly difficult for a node to remain in the same state, and eventually it switches its activity state to the complementary value. When such a network is trained with multiple input stimuli it will oscillate between activity states representing these stored memory patterns.

Recently the Horn and Usher model was extended to include a weight decay term so that the order of input pattern presentations could affect the network’s recall (Reggia et al., 2009; Winder et al., 2009). This allows the network to accurately model the recency effect observed in human working memory on running memory span tasks. Stimuli which were presented later in the input sequence were more likely to be successfully stored and recalled by the network when using weight decay.

While this memory model was able to match the position-specific recall rates of human subjects, the order in which the stimuli were recalled was arbitrary. In this chapter, the oscillatory weight decay network is augmented to enable it to recall inputs in the order presented. This is achieved by introducing a second set of temporally asymmetric weights into the model. By doing so the network is induced to oscillate between stored memory states in the desired order.

More specifically, in the work presented here temporally asymmetric Hebbian learning is used in oscillatory networks for the first time. Adaptation occurs in a fashion inspired by experimental evidence that synaptic efficacy in biological cortex and other brain structures is “temporally asymmetric” (Bi and Poo, 2001;

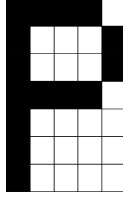


FIGURE 3.1. Stimuli to the model consist of 35 binary-valued inputs, conceived of as letters (such as the ‘P’ shown here) for ease of visualization and interpretation.

Markram et al., 1997; Zhang et al., 1998). That is, synapses are strengthened (LTP) if presynaptic activity precedes excitatory post-synaptic potentials by 20-50ms, and weakened (LTD) if the time course is reversed. The model presented here, when extended in this fashion, not only captures the recency effect of the original model (Winder et al., 2009) but also now largely retains the sequential order in which the stimuli were presented.

3.2 METHODS

3.2.1 MODEL DESCRIPTION

The model uses a fully connected network of N linear threshold units. Each node takes a binary value $a_i \in \{-1, 1\}$. The stimuli used are in effect arbitrary sets of N bits, although they are treated as being individual letters from A to Z for ease of interpretation. Figure 3.1 shows an input to a 35 node network interpreted as the letter ‘P.’

The operation of the model occurs in two phases: first a temporal sequence of input stimuli are presented and the weight matrices learned according to Equations 3.1 and 3.2 below, and then the model is allowed to oscillate between states according to Equations 3.3 and 3.5 for a predetermined total number of

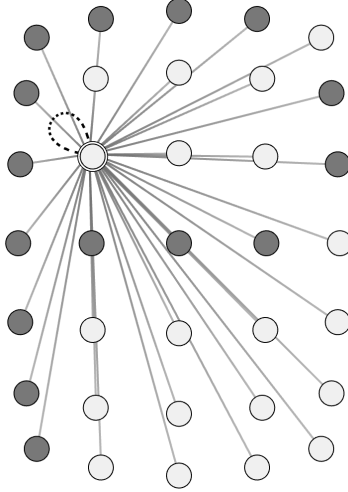


FIGURE 3.2. Architecture of the sequential associative memory. Only the connections of one node are shown. Nodes are interpreted as being arranged in a 5×7 grid; here they are offset so that depicted edges are not collinear to enhance legibility. Solid grey lines represent connections between nodes which have both symmetric and asymmetric weights. The dotted black line from the highlighted node to itself indicates that this self-connection has *only* asymmetric weights.

iterations. One iteration, or time step, corresponds to asynchronously updating every node once in random order.

3.2.1.1 TRAINING

There are two sets of connection weights, \mathbf{W} and \mathbf{V} . Both are $N \times N$ matrices composed of real values, and are initialized to zero before learning. The first of these, \mathbf{W} , is the same symmetric weight matrix used in previous version of this model (Winder et al., 2009). The entries of \mathbf{W} are updated as each stimulus is presented according to:

$$w_{ij}^t = (1 - k_d)w_{ij}^{t-1} + \frac{1}{N}a_i^t a_j^t (1 - \delta_{ij}) \quad (3.1)$$

where k_d is a decay rate ($0 \leq k_d < 1$), and δ_{ij} is Kronecker’s delta, which ensures that weights on self-connections are fixed at zero. This is, at its core, the same Hebbian weight change rule used in many previous neural network models. The difference is the addition of the decay term k_d that reduces the influence of older stimuli in favor of more recent ones.

The new element of this model is the incorporation of a second weight matrix \mathbf{V} . The purpose of \mathbf{V} is to allow the model to recall stimuli in the same order they were presented. In order to accomplish this, \mathbf{V} is trained with a temporally asymmetric learning rule

$$v_{ij}^t = (1 - k_d)v_{ij}^{t-1} + \frac{1}{N}a_i^t a_j^{t-1} \quad (3.2)$$

inspired by the learning method used in some past neural networks for processing temporal sequences (Schulz and Reggia, 2004). This is similar to the Hebbian learning with decay given in Equation 3.1, but it associates the activity of node i during the presentation of stimulus at time t with the activity of all other nodes j during the presentation of the *previous* stimulus at time $t - 1$ in the sequence. This introduces a sense of temporal ordering to the weight matrix, potentially making it possible to recall the stimuli in order rather than randomly as was previously done. Note that the decay term is still present, although the Kronecker’s delta factor is no longer used as it is desirable for a node’s activity to be influenced by its activation state in the previous time steps.

3.2.1.2 RECALL

After learning and before recall the network is initially set in a random activity state. It is not necessary to prime the network with a partial or noisy version

of any of the input patterns. The calculation of inputs to each node is modified from the prior model to account for both sets of weights. The input to node i at time step t is given as

$$h_i^t = \sum_j (\beta_W w_{ij} a_j^t + \beta_V v_{ij} a_j^{t-1}) - \theta_i^t \quad (3.3)$$

where the constant coefficients β_W and β_V are used to weight the relative contributions of \mathbf{W} and \mathbf{V} ($0 \leq \beta_W, \beta_V \leq 1$). θ_i is a dynamic threshold used to insure that the network oscillates between states rather than coming to rest at a fixed attractor. Its calculation has been simplified from previously, however, with it now being updated according to the following two rules. Every time step, θ_i decays according to $\theta_i^{t+1} = (1 - k_\theta) \theta_i^t$. In any time step in which the state of node i has remained unchanged from the previous time step a factor of $k_w a_i^t$ is also added to θ_i^{t+1} .

$$\theta_i^{t+1} = \begin{cases} (1 - k_\theta) \theta_i^t + k_w a_i^t & a_i^t = a_i^{t-1} \\ (1 - k_\theta) \theta_i^t & a_i^t \neq a_i^{t-1} \end{cases} \quad (3.4)$$

This moves θ_i in the direction of the activity state of node i , making it more difficult for node i to remain in the same state. Both k_θ and k_w are constants chosen in advance, with $0 < k_\theta < k_w < 1$. We use $k_\theta = 0.09$ and $k_w = 0.175$ in the following computational experiments, although similar values gave qualitatively similar results. Equation 3.3 has been simplified from the model it is derived from by dropping the K_i biasing term derived from Horn and Usher (1991). This was previously used to account for the potentially uneven distribution of active and inactive nodes across potential stimuli and current network state. Computational experiments revealed that it added computational complexity to the model without significant impact on performance.

After the input to each node is calculated, the node’s state is updated according to the following rule.

$$a_i^t = \begin{cases} +1 & h_i^t > 0 \\ a_i^{t-1} & h_i^t = 0 \\ -1 & h_i^t < 0 \end{cases} \quad (3.5)$$

This is also a simplification of earlier models, which used a stochastic updating process. We have found that the deterministic rule given above performs roughly the same with this and similar data sets and significantly reduces computational cost.

3.2.2 MEASURING RECALL

We assess the network’s recall by calculating the Hamming distance d_λ between its activity state \vec{a} and \vec{a}^λ , where \vec{a}^λ is a perfect representation of one of the 26 stimuli λ :

$$d_\lambda = \frac{1}{2} \sum_{i=1}^N |a_i^\lambda - a_i| \quad (3.6)$$

The greater the distance d_λ between \vec{a} and \vec{a}^λ , the lower the similarity $s_\lambda = 0.85^{d_\lambda}$ will be. A value of $s_\lambda = 1.0$ indicates a perfect match between \vec{a} and \vec{a}^λ . We call any such time step a “recall peak” for λ . An exponential function was used to define s_λ in order to emphasize the difference between some pairs of inputs with small Hamming distance between them. The choice of 0.85 in the definition of s_λ is essentially arbitrary, chosen because it produced visually reasonable results. Values such as 0.7 or 0.9 work just as well.

In order to compare versions of the model as to whether they successfully recalled the stimuli in the same order as they were presented, a record is made of the transitions from one recall peak to the next and this is used to generate a single scalar value. We count the proportion of these peak-to-peak transitions which occur between one stimulus and the stimulus which was presented to the network immediately following. A transition from the fourth-back to the third-back stimulus would be counted as a correct transition, while one from the third to the fourth, or fourth to second, would not. A higher proportion of such correct transitions is indicative of the recall being more well ordered in the sense that the model is cycling through the stimuli it recalls in the same order as they were initially presented. Transitions following the one-back stimulus (i.e., the final stimulus) are ignored because there is no “next” stimulus to correctly transition to.

The recall phase of the model lasts for hundreds of time steps, each one potentially generating the recall of a stimulus. This lengthy series of activity must be distilled into a single ordering of the inputs, in which each unique stimulus appears no more than once. This is accomplished by consolidating any consecutive time steps in which the network peaks for the same stimuli. (Neither human subjects nor the model were ever presented with duplicates of the same stimulus, so there was no cause for the model to report seeing the same stimulus repeated.) So, for instance, if a stimuli sequence of ‘ABCDE’ were to result in the network oscillating between the states ‘BCCDDE’ then the recalled sequence would be taken to be ‘BCDE,’ and the second through fifth stimuli would be considered to have been remembered correctly. The requirement to remember the stimuli in the appropriate position is the same as what human subjects are faced with when doing running memory span tasks.

Previous versions of the model were not subjected to this requirement; any recall peak for a stimulus was enough for it to be considered correctly stored.

3.2.3 HUMAN BEHAVIORAL DATA

Previously collected human behavioral data (Winder et al., 2009) on the Running Memory Span task was used for comparison with the new model's performance, following the designs of Pollack et al. (1959) and Bunting et al. (2006). The data was obtained from 38 adult subjects, all of whom completed the task satisfactorily. They were shown a rapidly presented, two per second sequence of 12 to 20 randomly ordered stimuli under computer control, and were asked to remember the most recent six items in the order of their presentation. Subjects indicated the stimuli that they recalled by clicking on a subsequent graphical display of all possible stimuli. Recall was measured by assessing accuracy of recall as a function of stimulus position. A stimulus was counted as accurately recalled only if: *(i)* it was presented in the retention window (e.g., the last six items, depending on instructions), *(ii)* it was correctly recalled by the participant; and *(iii)* it was recalled in the same position as it was presented, counting backwards from the final, most recent stimulus. Any item presented prior to the retention window that was recalled was considered a false positive, as was any item that was not presented at all but which was recalled. Any item from the retention window that was not recalled was considered a miss. Any item that was presented in the retention window, but which was recalled in the incorrect position was also counted as wrong (e.g., if the last six items presented were ABCDEF and the subject recalled DCBF EA then only E was counted as correct). A total of twelve trials were conducted for the task

with each subject requiring roughly 20 minutes per trial; no time restrictions were placed on subject responses. All 38 subjects completed the task.

3.3 RESULTS

The model’s recall was evaluated on the basis of both accuracy and peak-to-peak transitions occurring in the correct order, as described in the previous section.

In addition to comparing the model’s performance to human subjects, it was also compared to a previous model for this task (Winder et al., 2009) which uses dynamic thresholds but not asymmetric weights. Figure 3.3 shows an example of the effect that introducing asymmetric weights has on sequential recall. A plot of peaks in similarity for each of the stimuli presented is shown. In Figure 3.3(a) recall occurs without temporally asymmetric weights. As a result the ordering of the peaks is largely random, with the network moving between four stored memory states without regard to their original presentation order. In contrast, Figure 3.3(b) shows recall with asymmetric weights. Recalled memory patterns are much more ordered in their progression, with activity tending to proceed from earlier to later input patterns. This ordered retrieval of stored memories is much closer to the human behavioral task described above than was the earlier model which used only temporally symmetric weights.

Table 3.1 shows the number of stimuli successfully stored and recalled by the network for various values of β_W and β_V when the network is presented with a sequence of six inputs. In constructing Table 3.1, five hundred random sequences were used for each simulation, and the network was allowed to oscillate for 250 time steps, with $k_d = .15$. The cell corresponding to $\beta_W = 1.0$, $\beta_V = 0.0$ is

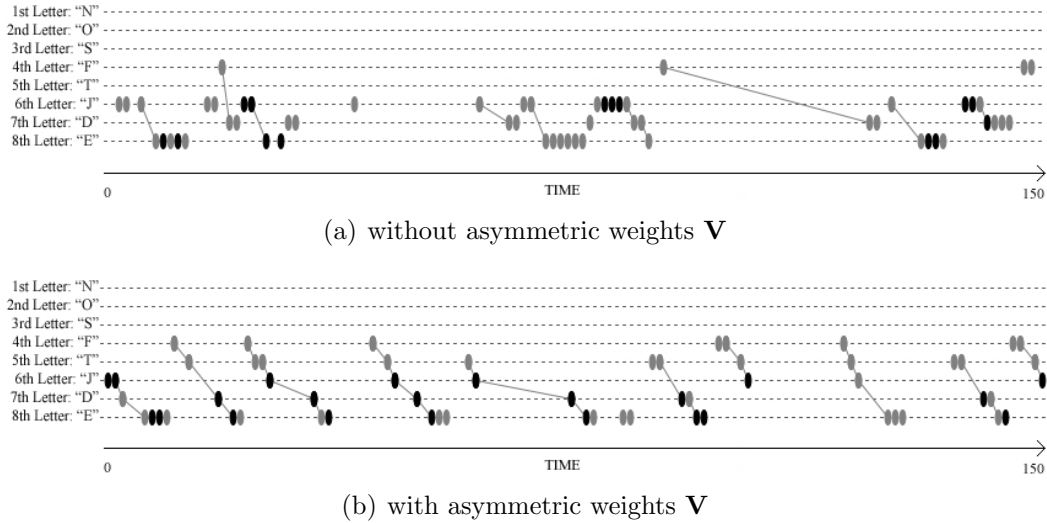


FIGURE 3.3. Plot over time of when the values of similarity s reached their peaks for the eight stimuli during an example run of the model. Black marks indicate when s reached the maximum possible value of 1.0 and thus were counted as present, while gray marks indicate when s exceeded 0.8 but did not reach 1.0. The lines between activity peaks indicate transitions that occurred in the same order as the stimuli were presented. The first 150 time steps of the recall phase are shown here. Figure 3.3(a) is without asymmetric weights ($\beta_W = 1.0$, $\beta_V = 0.0$), and Figure 3.3(b) is with asymmetric weights ($\beta_W = 0.5$, $\beta_V = 1.0$). In the former, one can see that the oscillatory states alternate between the four recalled memory patterns for the 4th, 6th, 7th and 8th stimuli (F, J, D and E). Note that these peaks largely occur in an arbitrary order. In the latter case, the network state alternates between the five most recent stimuli, i.e., it has a propensity to recall input stimuli in the same sequence as that in which they were presented.

equivalent to running the network without any influence from the asymmetric weights. The best results were achieved with $\beta_W = 0.5$, $\beta_V = 1.0$, which gave a capacity of 2.26 items and with $\beta_W = 0.25$, $\beta_V = 0.75$, which gave 2.22 items. For comparison, human subjects had a memory capacity of 2.73 items.

Furthermore, asymmetric weights increase position-specific recall performance in addition to increasing the total memory capacity relative to baseline ($\beta_V = 0$). Figure 3.4 shows the recall rate at each stimulus position for networks

TABLE 3.1. Number of stimuli recalled.

		β_V				
		0.00	0.25	0.50	0.75	1.00
β_W	0.00	—	1.13	1.38	1.46	1.54
	0.25	1.18	1.84	2.01	2.22	2.12
	0.50	1.44	1.91	1.89	2.04	2.26
	0.75	1.72	1.88	1.95	2.02	2.08
	1.00	1.76	1.90	1.93	1.93	1.85

TABLE 3.2. Proportion of peak-to-peak transitions in correct order.

		β_V				
		0.00	0.25	0.50	0.75	1.00
β_W	0.00	—	0.81	0.86	0.93	0.87
	0.25	0.56	0.71	0.71	0.83	0.78
	0.50	0.50	0.70	0.68	0.79	0.85
	0.75	0.56	0.65	0.68	0.75	0.78
	1.00	0.53	0.61	0.67	0.74	0.71

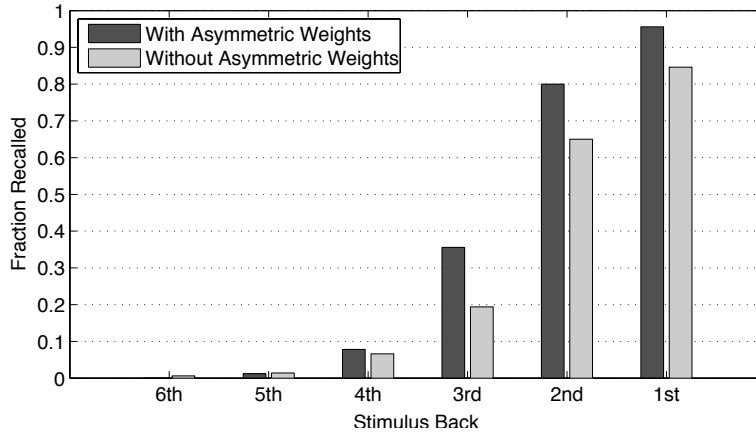


FIGURE 3.4. Recall rates for each position with and without temporally asymmetric weights. Five hundred random stimuli sequences were run using a decay rate of $k_d = 0.2$. Networks with asymmetric weights enabled used $\beta_W = 0.5$, $\beta_V = 1.0$.

both with and without asymmetric weights. Asymmetrically weighted networks were significantly more likely to retain the three most recent inputs.

Figure 3.5 shows that the network is capable of modeling human recency behavior on Running Memory Span when using asymmetric weighting by properly tuning the decay parameter, β_W and β_V . The model provides close matches for human performance on both 6-back and 12-back running span tasks. (For the former $k_d = 0.05$, $\beta_W = 0.5$ and $\beta_V = 1.0$ and for the latter

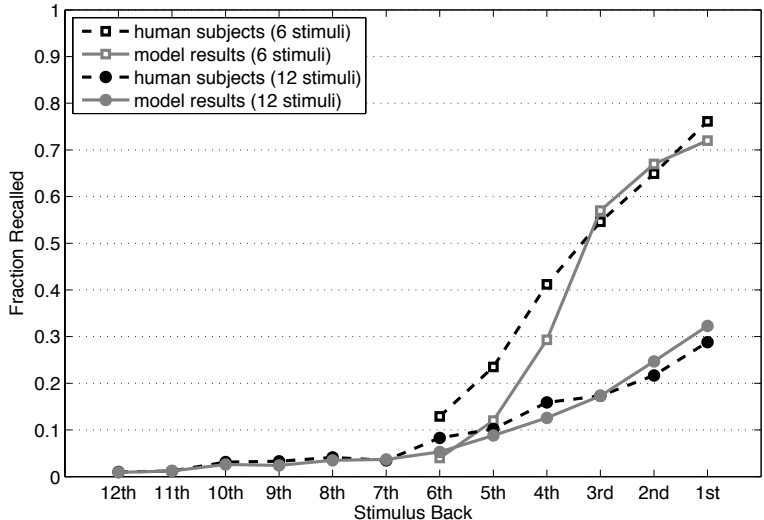


FIGURE 3.5. Comparison of the position-specific fraction of recalled stimuli by the model and human subjects for both 6-back and 12-back tasks.

$k_d = 0.075$, $\beta_W = 0.63$ and $\beta_V = 0.37$) Fitting data derived from human subjects is a simple matter of tuning these three coefficients, which was accomplished here with a simple iteratively-refined grid search, minimizing the RMSE.

In addition to having higher total and position-specific capacity, asynchronous weighted networks also retained the ordering of the input sequence more effectively. Table 3.2 gives the proportion of peaks in similarity s that occur in the correct order, using the same parameters as Table 3.1. That is, those that progress from the fourth-back to the third-back, for example. A high proportion of such transitions is achieved when the synchronous weights are ignored completely (i.e., when $\beta_W = 0$), but note that the number of stimuli recalled by such networks is significantly lower. The fewer items stored at all, the easier it becomes to get them into the correct sequence. Limiting the results to those networks which stored more than two of the six stimuli on

average, we again find that $\beta_W = 0.5$, $\beta_V = 1.0$ gives the best result with 85% of the peaks in s transitioning correctly, compared to between 50 and 56% for the fully temporally symmetric networks, regardless of β_W .

3.4 DISCUSSION

This chapter adds to the growing range of current models of short-term memory. It explains some of the richness of human memory behavior, for instance the limited memory capacity and recency effect in sequential recall tasks, but does so while remaining parsimonious in its design. There is no need to explicitly specify lateral inhibition in order to provoke competition between stored patterns, such as in Haarman and Usher (2001). In contrast, competition is allowed to arise from the process of Hebbian learning and dynamic thresholds. Further, we do not use different structures for different phases of the memory process. There is no complex architecture of learning and recall units, or structures to explicitly guide the recall process (Frank et al., 2001; O'Reilly and Frank, 2006). Rather, a single substrate of identical nodes is all that is needed. The two weight matrices used in the model are also trained with nearly identical rules, and are treated identically during recall. There is also no need to introduce extra layers or nodes to provide temporality of network activity, or to introduce recurrent connections or back-propagation between layers (Botvinick and Plaut, 2006). Multiple patterns, along with their order of appearance, can be stored on the same neural substrate simultaneously.

For the limited range of data considered here, this model did not need to maintain a unified record of the entire sequence of stimuli. Correlations between temporal events can be reconstructed by the network during recall in

order to preserve the entire sequence, despite the network only being aware of the immediately preceding stimulus during training. The model’s temporal “awareness,” such as it is, only exists in a thin temporal slice. Similarly, during the recall phase, each change of a node’s activity is only dependent on the immediately preceding state of the network. Of course, with more complex data additional processing mechanisms may be needed.

While this model appears similar to “chaining,” it is important to recognize that it does not suffer from one of the principle weaknesses of chaining as a technique for storing sequences: that a single error in recall will break the thread connecting consecutive items, causing the model to be unable to continue with the rest of the sequence. My approach, like that of Botvinick and Plaut (2006), SOB (Farrell and Lewandowsky, 2002), and TODAM (Li and Lewandowsky, 1993), avoids this issue by using distributed encodings of items on a single substrate and embracing noisy, stochastic processing. To borrow an analogy from Li and Lewandowsky (1993), if the memories in traditional chaining are like beads on a string, GALIS’ sequence memory is like superimposing several photographs on a single frame of film. A small error would break the thread, causing the rest of the beads to be lost, but a small piece of the film being damaged does not significantly impact the rest of the memories. The inherently stochastic nature of the network’s activity means there is little harm in being “knocked out” of sequence as the model is able to pick up the trail again (Lewandowsky and Farrell, 2003). (In fact, the initial state of the network is already out of the desired sequence. It is initialized to a random pattern, and not a noisy or partial version of the first pattern in the sequence or a special start-of-list marker, as is common.) From this initially random state the network is able to progress through the sequence, occasionally going astray,

but even then tending back towards the proper ordering because the errors occur in only some of the nodes, not an entire item representation as would be the case in a localist model.

Note that other difficult conditions for chaining, such as duplicate stimuli, repetitions, and interleaving confusable and non-confusable items, were not present in the tasks that human subjects performed, and so were left out of the model's training as well.¹ It would be instructive to test these conditions in the future. Despite the challenge they present, work such as Botvinick and Plaut (2006) shows that recurrent neural networks as a class are capable of handling such situations. Furthermore, Botvinick and Plaut partially attributed their success on those difficult conditions to the way their model encoded each item independently of the way other items were encoded. The attractors in my model of sequential recall maintain a similar independence which I believe may allow them to capture some of the same behavior, although this hypothesis is untested.

¹ See Baddeley (1968) and Henson et al. (1996) for discussion of conditions which are difficult to account for using only chaining models.

Learning Instruction Sequences for Control

4.1 INTRODUCTION

The asymmetric weights approach for sequential memory presented in the previous chapter expands the capabilities of attractor networks in a valuable way, and in doing so captures some interesting properties of sequential working memory. The goal of this chapter is to build a cognitive model which uses this technique to not only learn sequences of stimuli from the external environment but also to learn sequences of instructions required to perform its task. To accomplish this, multiple networks of the type described in the previous chapter are linked together to form a network of “regions and pathways” that is controlled using gated connections—that is, by using connections between regions whose behavior can be controlled by a third network. In addition, the associative memories discussed in the preceding chapter are enhanced to allow them to store multiple sequences in the same substrate or region concurrently. This approach, termed GALIS for “Gated Attractors Learning Instruction Sequences,” is demonstrated in this chapter by building models

for two different tasks: one called Store/Recognize, of my own design, and one called n -Back, which is commonly used in cognitive psychology.

The model of the prior chapter successfully captures many aspects of human working memory. However, it still relies on exogenous control: the occurrence of all weight changes, updates, recollections, accuracy assessments, etc., are controlled by the modeler, not by the model itself. In order to make control internal to the model I have linked a number of such attractor networks together, in addition to other regional modules, using gating. This expanded system uses sequential attractor networks not just to learn memories of perceptual stimuli, but also to learn memories of the steps needed to perform tasks.

In order to make the algorithm for a task more tractable it is beneficial to be able to decompose it into multiple, smaller subroutines. Each of these subroutines then has its own sequence of instructions used to execute it. This allows for more modular algorithms but introduces the (non-trivial) requirement that a sequential attractor network store more than one sequence at a time. This is accomplished by a modification to the learning procedure given in Equations 3.1 and 3.2: a separate set of weights is learned for each constituent sequence in the same way that the weights were learned in prior versions for a single sequence. The weights are then averaged together so that only a single set of weights is needed no matter how many sequences are being stored. This independent training of each sequence ensures that both the weight decay and the asymmetric weights operate only within a sequence and not between them. (For instance, the first element in the second sequence will not cause decay in the storage of the last element of the prior sequence.)

This leaves the problem of controlling which of these several sequences is recalled. To address this, the network is conceptually divided into two

partitions. (It is still fully connected, but the states of each set are interpreted as representing different things. See Figure 4.1.) Which of the trained sequences is recalled can be controlled by adjusting the input to one of these sets to provide context information to the network. The nodes used to provide this context are termed the “cue” nodes. They have values associated with each sequence so the network can reproduce it when necessary. The other nodes are termed the “response” nodes; they are used to encode the items the sequences are composed of. This arrangement makes it possible to form memories of multiple sequences concurrently, even if sequences share elements in common.

This last point is crucial if such networks are going to be used in an executive control system. The response to multiple situations may require some of the same steps be taken, so it is important that learning multiple sequences not rely on them having disjoint sets of elements (*cf.* Botvinick and Plaut, 2002). For example, imagine a system which has been trained to prepare cups of coffee and tea. If the system was midway through making a cup of coffee and had just added sugar, you would want it to avoid following this by adding lemon, the next step in preparing tea. This is a danger because because the addition of sugar is a step in both procedures.

The rest of this chapter is organized in to three sections. The first of these covers a GALIS model for the Store/Recognize task, which I designed to serve as a introduction to how GALIS approaches cognitive control. The next section expands on this to present a model for n -Back, which is a popular task in cognitive psychology, and compares results from this model to those from human subjects. The chapter concludes with a discussion of GALIS and cognitive control.

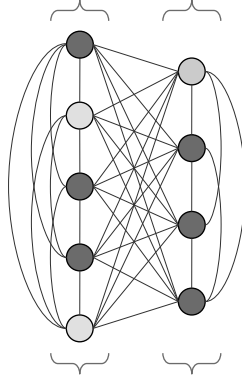


FIGURE 4.1. A depiction of a memory divided into two conceptual partitions. In the previous chapter, as well as most of the literature, auto associative memories are depicted in a rectangular grid so that we may easily visualize their contents pictorially. In truth, nodes have no locations; any arrangement is equally valid. The layout of nodes above emphasizes the “divided” nature of the memory introduced in this chapter to store multiple sequences. Its nine nodes are still fully connected, and could be interpreted as representing a 3×3 bitmap, or the binary string $101100111_2 = 359$. (Darker nodes = 1; lighter nodes = 0; reading top-to-bottom, left-to-right). However we can also interpret it as representing two different numbers: $10110_2 = 22$ (left column) and $0111_2 = 7$ (right column). In this way a network with a single, fully-connected set of weights can simultaneously be seen as forming an autoassociative memory of long patterns *and* a heteroassociative memory of two shorter patterns. I exploit this throughout the remainder of this dissertation to use one set of nodes to represent which sequence is being recalled, and the other to represent the items in that sequence.

4.2 THE ‘STORE & RECOGNIZE’ TASK

Store/Recognize is a task designed as the first test of the GALIS control system (Sylvester et al., 2011). This task is a first attempt to establish the basic idea of the GALIS control system works, before moving on to real-world problems and tasks from cognitive psychology. Store/Recognize was designed to be straightforward and easy to understand while still demonstrating some important features. Specifically, the Store/Recognize task was designed so

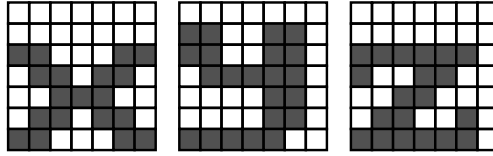


FIGURE 4.2. Three of the visual input patterns V used for the Store/Recognize task. The inputs are patterned after the IBM CGA typeset, as seen in Chartier and Boukadoum (2006). Light and dark squares denote a values of -1.0 and 1.0 , respectively.

that the control system would have to consider two different inputs, make two different decisions based on them, one after the other, add inputs to working memory, and search the contents of memory for a given pattern and recognize when it is present. In addition, the general process the model uses to address this task is roughly the same as that needed to address the n -Back problem discussed later in this chapter and the Card Matching task, discussed in Chapter 5. This task provides a good first illustration of how GALIS works that will facilitate understanding how these more complex tasks are addressed.

The Store/Recognize task consists of a series of visual inputs S , each paired with an instruction M to either commit the stimulus to working memory or to evaluate whether that stimulus is already in memory. These are termed the “load” and “evaluate” modes. (For the purposes of this task, M is treated as an input, but from a wider perspective it is a piece of contextual information about the task to be carried out being received from elsewhere in the brain, not from the external world like S .) Each visual stimulus is a bipolar pattern of length 49, which for convenience and ease of interpretation are visualized as a 7×7 grid of pixels with each stimulus again taking the form of a lower case English letter. See Figure 4.2 for examples.

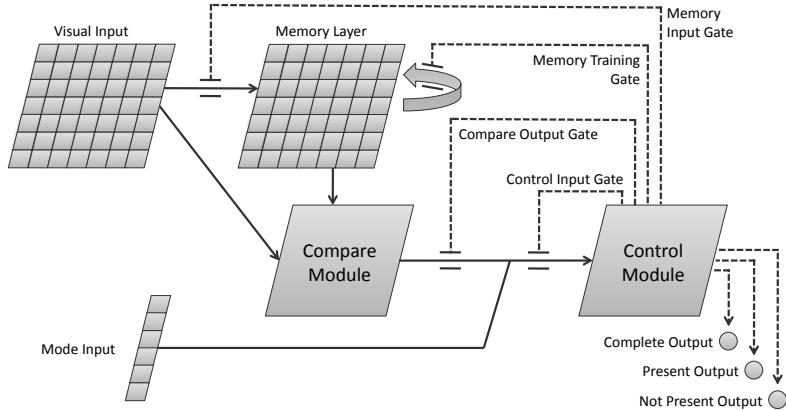


FIGURE 4.3. The GALIS model for the Store/Recognize task. Thin, solid arrows denote one-to-one connections. The recurrent connections of the memory layer are fully connected. Dotted lines are the outputs of the control module. Note that the number of boxes in the each layer is an approximation only, and does not faithfully represent the number of nodes used in the model. Details of the Compare and Control Modules can be seen in Figures 4.4 and 4.7.

After receiving a visual stimulus and a mode (S, M) , the model processes the stimulus until producing an output by activating one of three output nodes (Figure 4.3). When $M = load$ the model should activate the *complete* node, to signify that it is done storing the stimulus. When $M = evaluate$ the model should output either *present* or *not present*, depending on whether the stimulus has previously appeared. In the latter case, the model should also store the stimulus in memory before signaling its output. This means that when a pattern which has not previously been presented is evaluated twice, the correct output is *not present* the first time, but *present* the second, so output is dependent not just on the current input but also on input at previous times.

Output is produced only when the model has finished processing the input (S, M) , rather than at every time step. It may take a varying number of time steps from initial observation of an input until an output is produced, depending on the particular input, the prior state of the model, and the efficiency of

the model’s performance. For instance, more time steps are needed when $M = \textit{evaluate}$ and a visual input is not found in memory than when it is found, since the former requires storing the new visual input while the latter does not. Once an output is produced a new pair of inputs is presented and processing continues.

4.2.1 MODEL OVERVIEW

The GALIS model for the Store/Recognize task is composed of six components, which can be seen in Figure 4.3. They are the visual input layer, the mode input layer, the output nodes, the memory layer, the compare module, and the control module.

Nodes in the visual input layer take values in $\{-1, 1\}$, and are set externally to represent the visual stimulus V being presented in the current stage. As such, the layer consists of 49 nodes.

The mode input, also set externally, encodes the current system goal. Rather than using a local representation with a single node or pair of nodes to differentiate *load* from *evaluate* inputs, a bipolar pattern of length twenty four is used to represent each mode. This more accurately reflects the fact that this input would be provided to the control module from another brain region, and such connections use coarse, distributed representations rather than localized ones. The specific patterns used to represent *load* and *evaluate* are random bipolar patterns chosen in advance.

Three linear threshold units are used for output. There is one each for *complete*, *present*, and *not present*. For each of these, an input $x_i \geq 1$ will produce an output of one, otherwise output is zero.

The memory layer is a discrete Hopfield network forming an auto-associative memory. Hopfield networks are often used for long-term memories, but the memory layer of this model represents working memory, with limited capacity, high plasticity, and close integration with executive systems. The working memory nodes are bipolar valued, and the size is the same as that of the input layer. Training of the memory layer is accomplished with standard one-shot Hebbian learning with weight decay

$$w_{ij}^t = (1 - k_d)w_{ij}^{t-1} + \frac{1}{N}a_i^t a_j^t (1 - \delta_{ij}) \quad (4.1)$$

where k_d is a decay rate ($0 \leq k_d < 1$) and δ_{ij} is Kronecker’s delta, which ensures that weights on self-connections are fixed at zero (Winder et al., 2009). The weight matrix for the memory layer is termed \mathbf{W}_M , to differentiate it from the weight matrices needed within the control module, \mathbf{W} and \mathbf{V} . Note that the Store/Recognize task only requires remembering the set of stimuli which have been seen, *not* the order they were seen in. As such only a single, symmetric weight matrix is used to store the memory of external stimuli. In contrast, both symmetric and asymmetric weights are used in the memory of instructions since these must be recalled in order.

Input to each node i in the memory layer is composed of the influence of all other nodes in the layer along with a gated connection from the topologically corresponding node in the input layer:

$$h_i = 2g_{in}in_i + \sum_j w_{ij}a_j \quad (4.2)$$

where g_{in} is the value of the gating node mediating the input-to-memory connections, in_i is the state of node i in the visual input layer, w_{ij} is the strength of the connection from node j to node i both in the memory layer,

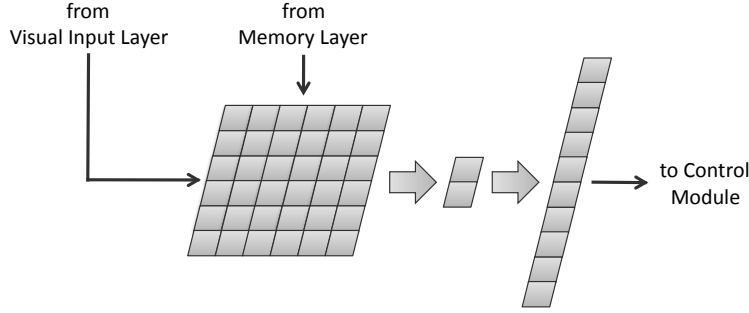


FIGURE 4.4. The compare module. Thin, solid arrows denote one-to-one connections. Thick arrows denote full connections.

and a_j is the state of node j in the memory layer. State updates to the working memory are the same as Equation 3.5 in the previous chapter.

$$a_i^t = \begin{cases} +1 & h_i^t > 0 \\ a_i^{t-1} & h_i^t = 0 \\ -1 & h_i^t < 0 \end{cases} \quad (4.3)$$

The visual input-to-memory connection is used to enable the memory to be influenced towards or away from the current stimulus. When the gate is fully open ($g_{in} = 1$), the state of the memory is forced to become the same as the input layer. When the gate closes ($g_{in} = 0$), there is no influence from the input layer and the memory layer operates as a standard auto-associative memory. The factor of two is necessary to ensure that the inputs coming through the gate are able to overwhelm the influence of the intralayer connections and effectively force the state of the memory.

The fifth component of the model is the compare module, which is used to judge the similarity between the current state of the input and the memory layer. It is composed of three layers, as seen in Figure 4.4. The first layer is the same size as the memory and visual input layers, and receives one-to-one

connections from each of those components. The state of nodes in this layer is the product of the states of the corresponding nodes in the input and memory layers.

The second layer of the compare module has two nodes, both of which take as input the sum of the nodes of the first layer, divided by the size of the input layer. (This produces a value proportional to the inner product of the input and memory layers' states.) One of the second layer nodes adopts a state of one if its input is above a certain threshold — for this model and set of inputs a value of 0.9 is used — and zero otherwise, while the other node outputs one if its input is below that threshold and zero if it is above. The state of this two node layer is then multiplied by a fixed weight matrix to produce a 32-bit bipolar pattern which serves as input to the control module. The patterns output by the compare module are static and pre-defined random ones, like the *load* and *evaluate* patterns.

The compare module outputs these 32-bit patterns for two reasons, one theoretical and the other practical. From a theoretical standpoint, the brain rarely uses local encodings to transmit data between regions, and so it is more plausible to design the model to use a distributed representation of the compare module's output. Regarding practicality, the patterns the control module uses internally are on the order of several hundred bits long. It aids the necessary mapping from the compare module's output to the control module patterns if the former is already thirty two bits long rather than two. In addition, the control module must learn to recognize whether the compare module is indicating a high or low similarity between working memory and visual input. This discrimination is easier in higher dimensions.

The final component is the control module, which is used to direct the operation of the rest of the model. It takes inputs from the mode input layer, and compare module. The control module has three outputs to the *complete*, *present*, and *not present* nodes, as well as four other outputs, called “gate control signals,” which operate gates in the rest of the model. For example, one of the control module’s outputs manipulates the gate between the input layer and memory layer, controlling how much the former influences the latter.

The core of the control module is a second discrete Hopfield attractor network, called the “Instruction Sequence Memory” (ISM). Unlike the working memory attractor network mentioned above, this network is modified to use temporally asymmetric weights, allowing it to store sequences of patterns. (Refer to Chapter 3 for more details. Call the symmetric weights \mathbf{W} and the asymmetric weights \mathbf{V} .) Each stored pattern corresponds to a particular action the network may take, or more specifically, to a particular set of signals to open and close different gates to different degrees. A sequence of these actions taken together corresponds to the steps needed to address a particular situation. For instance, when presented with a new visual stimulus when $M = load$, the model must take three actions sequentially: first, make the state of the memory layer match that of the visual input layer by fully opening the input-to-memory layer gate, then update \mathbf{W}_M to store the new pattern in memory, and finally output *complete* and prepare the model for a new input.

The control module has two other sections besides this temporally asymmetric attractor network which are called the “encoder” and “decoder.” They serve to translate the inputs to the control module into the particular patterns stored in the instruction sequence memory, and then translate the response of the instruction sequence memory into the control module’s final outputs. This

pre- and post-processing is done primarily to mitigate the effects of noise and to enable the control module to convert between inputs, stored patterns, and outputs of different dimensions.

4.2.1.1 CONTROL OUTPUTS AND GATING

The control module manages the behavior of the model through a system of gates. The outputs of the control module are used to open and close these gates, which in turn modulate the flow of activity between different layers. In addition to the opening and closing of gates, the control module also activates the three output nodes.

The four gates which control the flow of activity throughout the model can be seen in Figure 4.3. They are:

1. a memory input gate between the visual input layer and memory layer, so that the memory's current state can be biased towards or away from the current stimulus;
2. a memory training gate which controls when the working memory updates its weight matrix \mathbf{W}_M ;
3. a compare output gate which modulates the output of the compare module, so that it is possible to notice or ignore the similarity between input and memory state, and;
4. a control input gate which controls the encoder, so that the control module can control when it updates its own state.

When a gate is open it allows information to flow through it like an open valve in a pipe. (In contrast to an open switch in an electrical circuit, which would

prevent flow.) A gate’s state is given by

$$g^t = k_g g^{t-1} + s^t \tag{4.4}$$

where k_g is a decay term, here equal to 0.5, and s^t is the current value of the control module output governing this gate. The effect of gates is multiplicative, such that their downstream activity is a product of their incoming activity and their current state g^t . Control module outputs have values in $[-1, 1]$, so with $k_g = 0.5$ gates will have states in $[-2, 2]$. This allows gates to have more nuanced effects than binary states of “open” and “closed.” Like Stewart and Eliasmith (2011), a gate can have an amplifying effect on its incoming value ($g^t > 1.0$), a damping effect ($0.0 \leq g^t < 1.0$), or an inhibitory effect ($g^t < 0.0$). Being able to use the same system for both attending to an input (i.e., amplification) and inhibiting that input is appealing, since the two effects have been described in the cognitive psychology literature as antipodal (Engle et al., 1995).

The exception to this is the memory training gate. Because updating the working memory weights is a discrete decision — \mathbf{W}_M is either updated or not in any time step — the working memory gate has a threshold. Its state g^t is calculated the same way, and \mathbf{W}_M is updated when $g^t > 1$ and not updated when $g^t \leq 1$.

In some of the situations listed there are many connections being mediated by the same gate. For instance, the connection between the input and memory layer is one-to-one. You may think of each of these 49 connections as having its own gate, with each gate having an identical value. The effect is the same as a single, “master” gate controlling all 49 connections based on a single output from the control module and so I adopt the convention of referring to the

parallel opening and closing of these 49 connections as if there was a single gate present.

4.2.2 MODEL OPERATION

The control module is trained before the task begins, so that the instruction sequence memory already contains the appropriate pairings of conditions and responses. The encoder and decoder are also trained before the task begins. Training the control module occurs only once for this task, and once the task begins its weights remain unchanged. The memory layer, in contrast, begins the task in a blank, untrained state, and has its weights updated multiple times as the task progresses.

During each step of processing the model goes through the following operations.

If the model activated any of the three output nodes in the previous time step, a new stimulus will be presented, otherwise the inputs from the previous step are retained. Next, the state of the main memory is updated according to Equations 4.2 and 4.3. Then the output of the compare module is updated to reflect the new state of memory and the potentially new state of the input layer.

Next, the control module's encoder is updated if the gate regulating it is open. If it is not open, the encoder input will be the same as the previous time step, and so its output will be unchanged, and thus the downstream layers in the control module will receive the same inputs as the previous time step. This prevents the control module from switching to a new sequence of actions before the previous sequence has concluded, as this gate is opened only at the

end of a sequence, setting the stage for the next sequence of actions to begin in the following time step.

The encoder's output, whether it is the result of new inputs from an open gate or not, is then fed in to the cue nodes of the sequences memory. The instruction sequence memory is then updated to produce the next response pattern. This response pattern is processed by the decoder to select a particular action, which then outputs the gate control signals which compose that action.

The newly produced gate control signals are used to update the gating values according to Equation 4.4. If the gate responsible for training the main memory is open (i.e., its state is greater than 1.0), then those weights are updated at this point and the weight update gate has its value reset to -1.0 . Finally, the three output nodes are updated according to their associated control signals. If any output is activated, all three have their values reset to -1.0 , and a new stimulus will be presented to the model during the next time step.

4.2.3 CONTROL MODULE OPERATION

There are four different situations to which the model as a whole, and thus the control module, must respond. Each has an associated sequence of actions which form the desired response. Note that there are many-to-one associations between situations and responses, that is, the same action may be a member of more than one sequence. A listing of the situations and their associated response sequences can be seen in Table 4.1.

The first situation occurs when the model is given a new visual stimulus and $M = load$. In that case the model must store the current input in memory and then activate the *complete* output node. This is accomplished by first

TABLE 4.1. Instruction sequences stored in control module’s ISM.

Sequence	Action
1. load stimulus	1. open input-to-memory gate, strongly biasing memory state towards the input
	2. train the memory layer (\mathbf{W}_M)
	3. output “complete”
2. evaluate whether stimulus is present	4. open input-to-memory gate, slightly biasing memory state towards input
	5. close the input-to-memory gate, removing the bias to the memory
	8. open the output gate of the compare module
3. stimulus present	6. output “present”
	6. output “present”
4. stimulus not present	1. open input-to-memory gate, biasing memory state towards the input
	2. train the memory layer (\mathbf{W}_M)
	7. output “not present”

fully opening the input-to-memory gate, forcing the state of the memory to match the input layer. Next the control module opens the gate which allows the memory module to update \mathbf{W}_M , storing the current pattern in memory. Finally, the “complete” output node is activated and the gate which allows the control module to update its input is opened. This series of actions is depicted visually in Figure 4.5.

The second situation occurs when the model is presented with a stimulus, $M = \textit{evaluate}$ and the control module is not receiving any input from the compare module. The first action in the associated sequence opens the input-to-memory gate. The added input to the memory layer nodes reshapes the attractor landscape so that the activity of the memory layer is biased in favor

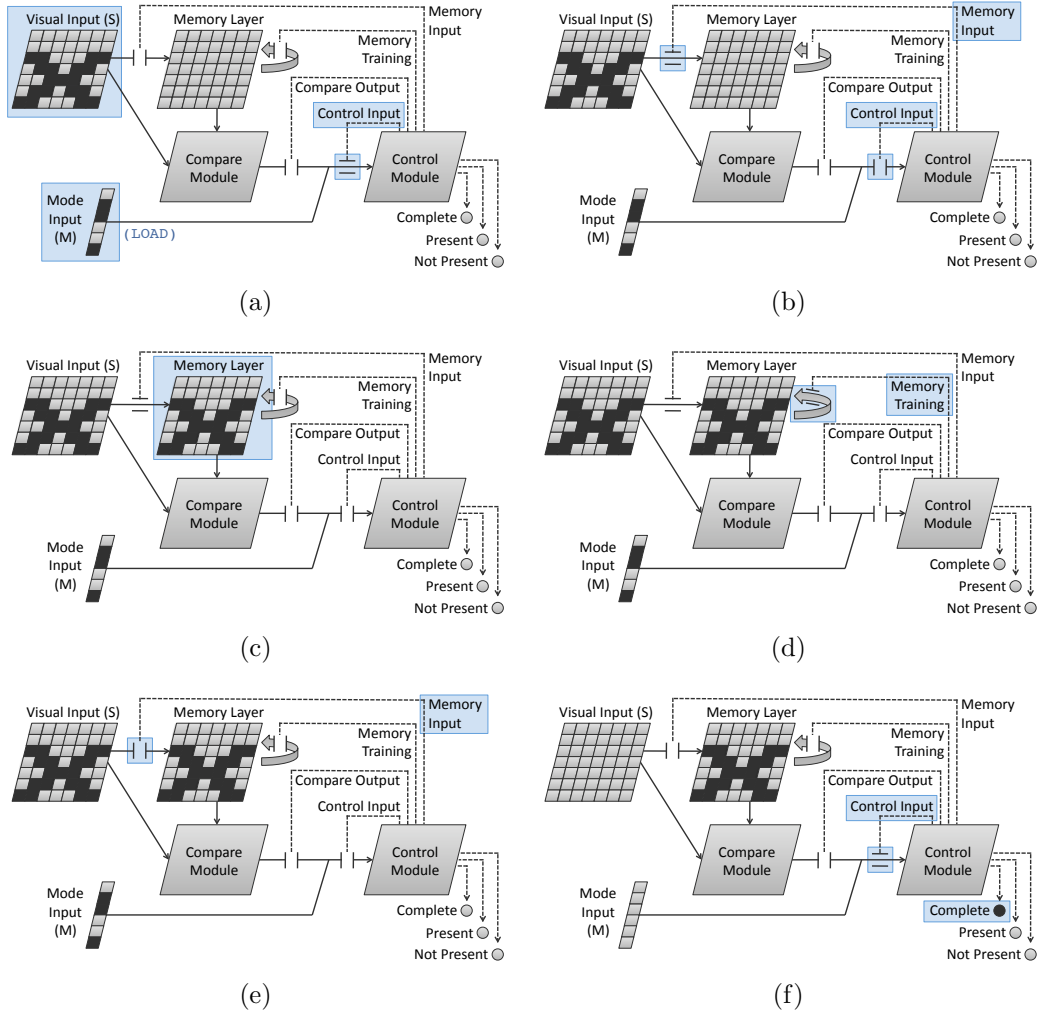


FIGURE 4.5. The series of actions to store a pattern in working memory. The process takes three time steps, but it has been unpacked here into six stages for clarity. (a) A new pair of inputs (S, M) is presented, and the control module input gate is open. Because $M = load$, the controller determines that it must execute sequence 1 in Table 4.1. (b) The first step is to open the memory layer's input gate. At the same time, the control module input gate is closed to prevent the controller from switching sequences before the current one completes. (c) Input from the visual layer causes the state of the working memory to match the visual input S . (d) To store S as an attractor state, the memory training gate is opened. (e) Now when the biasing input is removed S persists as the state of the working memory. (f) The *complete* output node is activated and the control module's input gate is opened so that it can select a new sequence of actions in the next time step.

of the pattern in the visual input layer. (Imagine the memory layer's attractor landscape as a rubber sheet, with the attractors being depressions in the sheet. The additional activity from the visual input layer is like a force pushing down on the sheet, creating an additional depression into which the activity state will fall.) The second action closes the input-to-memory gate, removing the bias from the memory module. If the input pattern has previously been stored in memory then the memory state should remain in that basin of attraction when the bias is removed since it will be in an energy minima. Conversely, if the pattern has not been previously stored, then when the memory is updated in the next time step the state should shift to a pattern which has already been stored. (See Figure 4.6 for a diagram of this process.) The final step of this sequence is to open the output gate of the compare module in order to judge whether the memory layer has remained in the same state or changed to a different one.

This sequence is followed by either the third or fourth sequence, depending on the output of the compare module. If the compare module indicates that the memory state and input layer match each other, then sequence 3 is begun, otherwise 4 is started.

Sequence 3 is simple, as all that needs to be done if the memory and input layers match each other (i.e., the input pattern has previously been stored in memory) is to activate the *present* node. There is, however, a small wrinkle. Note that response 6 appears consecutively in response to cue 3 in Table 4.1. This is an artifact of the instruction sequence memory. It has difficulty recalling degenerate sequences of length one, since there is no prior pattern to associate with in order to train \mathbf{V} . It is only necessary for the control module to output

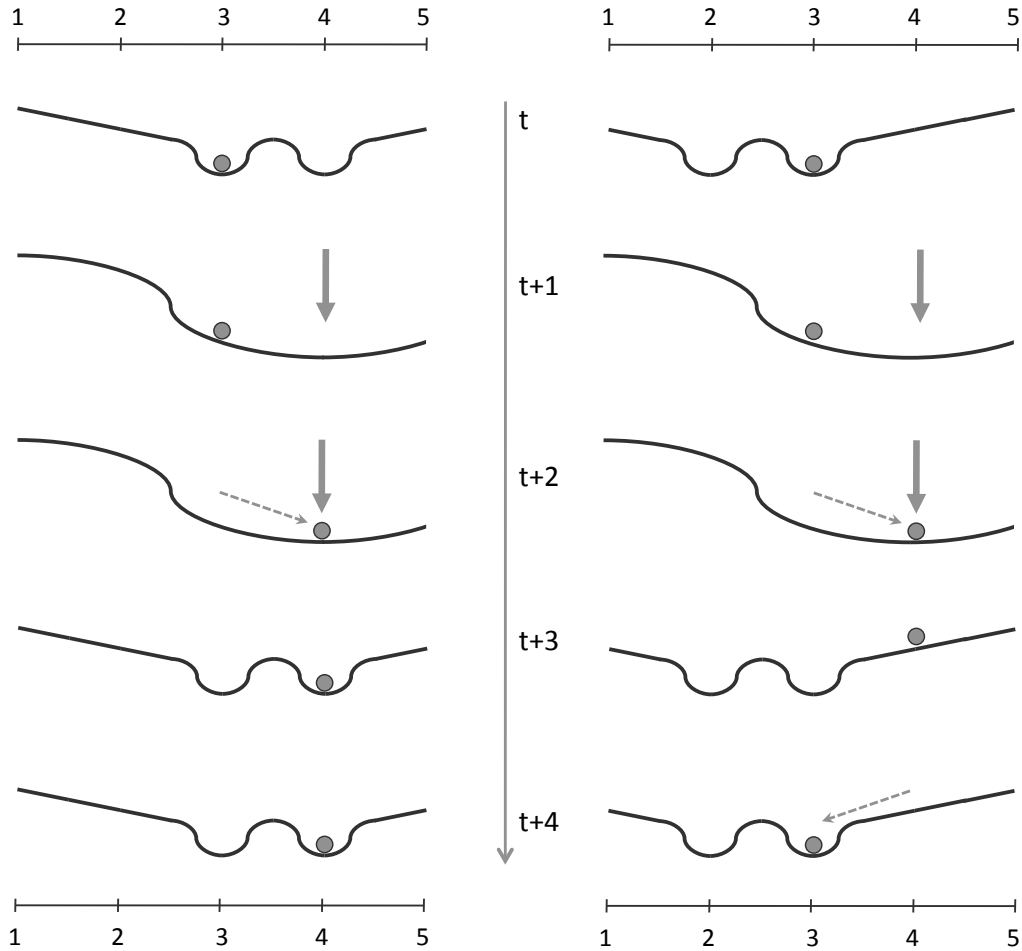


FIGURE 4.6. How biasing the attractor landscape is used to determine if a pattern is in memory. Each curve is a one-dimensional representation of an attractor landscape, with the state-space along the horizontal axis, and the effective energy along the vertical. On the left is an attractor landscape with basins of attraction at $x = 3$ and $x = 4$. On the right is another attractor with basins at $x = 2$ and $x = 3$. In both, the present state of the network, represented by a small circle, is initially $x = 3$. At time $t + 1$ both networks receive biasing inputs at $x = 4$, which deforms the attractor landscape as if the energy at $x = 4$ had been lowered. As a result at time $t + 2$ both networks change their state to $x = 4$. At time $t + 3$ the biasing input is removed, returning the attractor landscape to its prior condition. Because $x = 4$ is a basin of attraction in the network on the left, the state of that network remains at $x = 4$. However, $x = 4$ is not an attractor in the network on the right, so it shifts its state to $x = 3$. This allows GALIS to determine if $x = 4$ was a prior attractor state of the network: if it was an attractor then biasing the network at $x = 4$ temporarily will shift the network into that state even when the biasing input is removed, but if it was not an attractor then the network will not stay in the state after removing the input.

response 6 a single time in order to successfully activate the *present* output node, but it must be trained on a sequence consisting response 6 two times.

If the state of the memory does not match that of the input layer then the current input pattern has not been found in memory. In this fourth and final case the model must first store that pattern in memory, and then activate the *not present* output node. This occurs in the same way as the first sequence, though *not present* is output instead of *complete*.

4.2.4 CONTROL MODULE ARCHITECTURE

A diagram of the control module's internal structure can be seen in Figure 4.7. It is composed of three subcomponents. The principal of these is the instruction sequence memory which is used to store the actions needed to respond to each circumstance. The other two components are an encoder and decoder, which are used for pre- and post-processing to convert the inputs of the controller into the patterns stored in the instruction sequence memory, and then from those patterns into the signals the control module outputs.

4.2.4.1 INSTRUCTION SEQUENCE MEMORY

The instruction sequence memory is a discrete associative memory modified in two ways. The first is the incorporation of temporally asymmetric weights, which makes sequential recall possible, as shown in Chapter 3. The second is a conceptual division of the nodes into two sets, the "cue" and "response" nodes.

The role of the cue nodes is to provide the necessary context information to the response nodes to enable them to produce a series of outputs based on a single input. The state of the cue nodes corresponds to the situation the model is facing. The state of the response nodes in turn corresponds to one

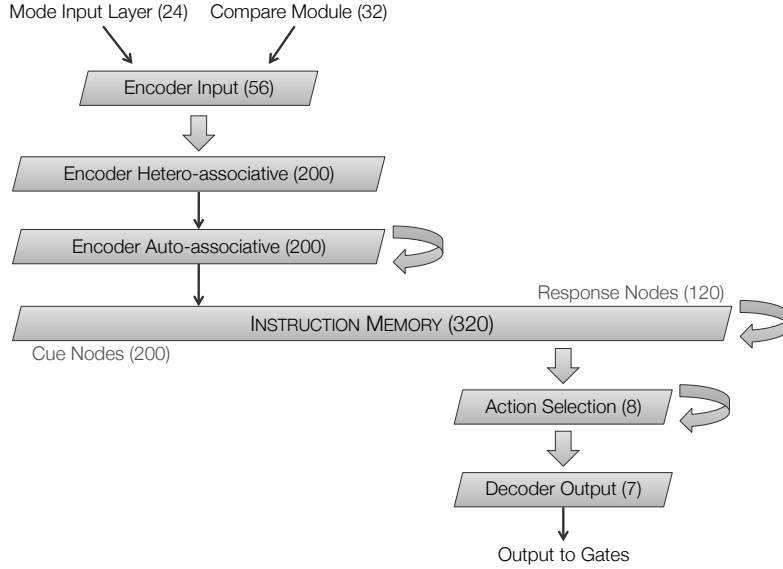


FIGURE 4.7. The control module. Thick arrows denote fully connected layers, while thin arrows denote one-to-one connections. The number following the name of each layer is the number of nodes it contains.

of the actions which should be taken for the current context. The network will be trained on multiple sequences, so the cue nodes provide the context information necessary to prompt the network to reproduce the correct sequence in the response nodes.

As an example, one pattern the cue nodes may take corresponds to having $M = evaluate$ and having the compare module indicating that the state of the main memory and input layers are approximately equal. In the event that cue is given, the proper response is to output the series of three response patterns which will lead to the model activating the *present* output node. (Sequence 3 in Table 4.1.)

There are four different cues and eight responses used for Store/Recognize. Because some responses are associated with more than one cue (that is, the same action is required in more than one circumstance), there are actually ten

total patterns stored in the instruction sequence memory. Each of these ten is the concatenation of a cue pattern with one of the responses required for it. This combined pattern, along with the others derived from the same cue, together form a sequence the memory is trained to recall.

Like the patterns chosen to represent the inputs and results of the compare module, the patterns chosen to represent each cue and response are random bipolar strings.

Although the instruction sequence memory is divided into cue and response groups the network is fully connected. The difference is that cue nodes receive an extra input from the encoder to bias their state towards the encoder's output. Additionally, only the state of the response nodes is read by the decoder. The GALIS model for Store/Recognize uses 200 cue nodes and 120 response nodes.

Learning in the ISM is again based on the methods successfully used in Chapter 3. Specifically, the ISM has two weight matrices, \mathbf{W} and \mathbf{V} , both of which are trained in advance of running the model on the memory tasks. The former is a typical weight matrix for an autoassociative memory, trained with one-shot Hebbian learning with weight decay, as defined in Equation 4.5.

$$w_{ij}^t = (1 - k_D)w_{ij}^{t-1} + \frac{1}{N}a_i^t a_j^t (1 - \delta_{ij}) \quad (4.5)$$

(Note that the coefficient for decay in the instruction sequence memory is denoted k_D , and is distinct from the decay term k_d used for the main memory in Equation 4.1, which can take a different value.) The second weight matrix, \mathbf{V} , also uses Hebbian learning but associates the state of a node not with the current states of other nodes, but with the other nodes' *previous* states. The

learning rule is given in Equation 4.6.

$$v_{ij}^t = (1 - k_D)v_{ij}^{t-1} + \frac{1}{N}a_i^t a_j^{t-1} \quad (4.6)$$

As mentioned in Section 4.1, in order to allow the instruction sequence memory to store multiple sequences, a change was made from the prior chapter regarding how \mathbf{W} and \mathbf{V} are formed from these individual weight updates. For each sequence l to be stored, a separate \mathbf{W}_l and \mathbf{V}_l matrix are generated according to Equations 4.5 and 4.6. Once all \mathbf{W}_l and \mathbf{V}_l matrices have been calculated they are then averaged together to get the final \mathbf{W} and \mathbf{V} matrices. This batch learning-like process is done because decay should only have an effect on intra-sequence ordering of patterns, not the order the sequences themselves happened to be trained in.

Updating the state of the instruction sequence memory occurs in two stages, the first governed by the asymmetric weights, and the second by the symmetric weights. This two-part update process is a change from the previous work of Chapter 3, in which the effects of both \mathbf{W} and \mathbf{V} were combined in a single input calculation. When both \mathbf{W} and \mathbf{V} are used simultaneously they can work at cross purposes. \mathbf{V} is pushing the network towards the next attractor, while \mathbf{W} is fighting to keep it in the same attractor basin. The new process helps the network to proceed from one state to the next in a much more orderly and predictable progression.

First the input to each node is calculated using \mathbf{V} and the previous network state

$$h_i^t = \sum_j v_{ij} a_j^{t-1} - \theta_i^t + e_i^t \quad (4.7)$$

where e_i is the input from the encoder ($e_i = 0$ when i is a response node), and θ_i is a dynamic threshold that is used to keep the network from settling into any one attractor basin. If a node's state has not changed in the previous time step, θ_i rises (if $a_i = 1$) or falls (if $a_i = -1$), which means node i will require inputs with increasingly larger magnitudes to remain in the same state. More specifically, at every time step, θ_i decays according to $\theta_i^{t+1} = (1 - k_\theta)\theta_i^t$. In any time step in which the state of node i has remained unchanged from the previous time step a factor of $k_w a_i^t$ is also added to θ_i^{t+1} . (This process is formalized in Equation 3.4 *supra*.) Here $k_\theta = .02$ and $k_w = 0.0125$.

The input h_i is then used to update the state of each node according to Equation 4.3. Using only \mathbf{V} to update the network serves to move the network state from the current attractor basin to the basin associated with the next pattern in the sequence.

After updating both \vec{a} and $\vec{\theta}$, the updating process begins again, this time using \mathbf{W} and the current network state, according to the following input rule:

$$f_i^t = \sum_j w_{ij} a_j^t - \theta_i^t + e_i^t \quad (4.8)$$

This helps the network to settle further into the new attractor basin it was pushed towards by \mathbf{V} in the previous stage. This new input is then used to update \vec{a} according to Equation 4.3 again (though the conditionals are predicated on f_i , not h_i this time). Finally, $\vec{\theta}$ is updated according to the rules given above once again.

4.2.4.2 ENCODER

The encoder is responsible for translating between the inputs to the control module — the output of the compare module and the mode input layer — and

the cue portion of the patterns stored in the instruction sequence memory. It is necessary to translate between them since the size of the input vector doesn't match the number of cue nodes. (There are 56 of the former and 200 of the latter.) By using more dimensions in the instruction sequence memory the stored patterns are more distinct from each other, and thus can be stored with less interference.

The encoder is composed of three layers of bipolar nodes, depicted in Figure 4.7. This architecture could possibly be made simpler, but only at the expense of more complicated dynamics. As is, each piece of the encoder has its own separate role for which it is specialized. Combining layers would disrupt this specialization of labor, which would in turn lead to interference between the functions being carried out.

There are 56 nodes in the first layer (one per input) and they receive input from outside the control module. The second and third layers have 200 nodes, one per cue node in the instruction sequence memory. The connections between the first and second layers are trained by way of standard one-shot Hebbian learning, which forms the two layers into a heteroassociative memory. The goal is to be able to produce the correct cue pattern when given the corresponding mode vector and compare module output.

The third layer is a standard autoassociative Hopfield network with one-to-one outputs to the instruction sequence memory cue nodes. It has a full set of intralayer connections which have been trained by one-shot Hebbian learning to recognize cue patterns. These connections serve to move the state further into the current activity basin, i.e., closer to the cue pattern that the heteroassociative process produced. This helps to mitigate errors in the result of the first two layers. By training only on the cue patterns and not the control

module’s inputs or response patterns the Hopfield network in the encoder specializes in cue pattern memory, and avoids interference which can arise in either the initial layers of the encoder or the instruction sequence memory. Combining layers of the encoder would eliminate this specialization of roles and reintroduce interference between these operations.

The encoder’s auto-associative memory layer is carrying out a redintegration process, which is the recovery of a pattern from a partial copy (Stuart and Hulme, 2009). Many cognitive models require that a pattern retrieved from memory be redintegrated in some way (Lewandowsky and Farrell, 2003), including CLARION (Sun, 2006) and the “clean-up” memory of Eliasmith and colleagues (presented in (Stewart et al., 2011) and used in (Choo and Eliasmith, 2010), among others). Attractor networks have been used for this purpose before, including Lewandowsky (1999) and Kesner et al. (2000).

As mentioned, the control module is capable of using gating to regulate itself. This is done by gating the inputs to the encoder. When this gate is open activity flows from the compare module and mode input layer to the encoder’s input layer, and is then operated on as described above. When the gate is closed the encoder does not receive input and the state of its input layer remains unchanged from the previous time step. This in turn means that the encoder’s output will be the same as the previous time step, so the instruction sequence memory will be operating on the same cue pattern as it did in the previous time step. This is done to prevent the instruction sequence memory from switching to the next sequence before the previous sequence is completed. The encoder input gate is opened as part of the final action of a sequence. This has the effect of allowing the control module to ready itself in the next time step to begin processing a new sequence of actions.

4.2.4.3 DECODER

The decoder is composed of two layers. The first, called the action selection layer, has eight binary nodes — as many as there are response patterns. It is fully connected to the response nodes of the sequences memory. These connections are trained by one-shot Hebbian learning to associate each response pattern with a single active node in the action selection layer.

In order to help ensure that only one node will be active, the action selection layer also has a set of recurrent connections. This sets up a competitive dynamic, with every node reinforcing its own activation while inhibiting that of the other nodes. Since the Hebbian weights from the response nodes to the action selection layer have already produced an activity pattern which is close to having a single winner, only a single step of these competitive dynamics is needed to make one node maximally active and all other nodes off.

The action selection layer then feeds in to the control module's output layer, which generates the actual control signals used to adjust the gates in the model. This is accomplished using a weight matrix which, in the current version of the model, is hand-coded in advance to produce the desired behavior. The outgoing weights from each action node are equal to the value that the corresponding control module output should have when that action is taken. For instance, if action i necessitates fully opening the first gate, partially closing the second, and leaving the others unchanged, action node i 's outgoing weights would be $(1, -0.25, 0, 0, 0, 0, 0)$.

TABLE 4.2. Inputs and desired outputs for a run of the first set of experiments.

Stimulus #	1	2	3	4	5	6
Visual Input	A	B	A	X	Y	X
Mode Input	load	load	evaluate	evaluate	evaluate	evaluate
Correct response	complete	complete	present	not present	not present	present

4.2.5 RESULTS

A series of computational experiments was conducted to evaluate this basic GALIS model on the Store/Recognize task. The first set of experiments uses random series of six inputs where the first and third as well as fourth and sixth visual inputs match, for example, ABAXYX. Details are in Table 4.2. Five hundred iterations using random visual stimuli were run. The model was judged based on the accuracy of each of the six outputs for each run, as shown in Figure 4.8.

The average accuracy across all responses in all trials was 90.6%. Predictably, the sixth input is the most difficult to respond to correctly, because it requires the stimulus to have been identified as not present on its first presentation, and thus stored in working memory, for it to be correctly identified as present later.

Varying the decay rate k_D in Eqs. 4.5 and 4.6 governing the control module's instruction sequence memory indicated (surprisingly) that small, *negative* decay rates performed the best. With a negative value of k_D previous items in a sequence are amplified rather than weakened. This amplification enlarges the basin of attraction for earlier patterns in a sequence, making it easier for the memory to run through a sequence from the beginning. With a very small, negative decay rate (-0.05), the average accuracy increased to 93.8%. The idea of a negative decay rate seems odd at first glance, but it is effectively

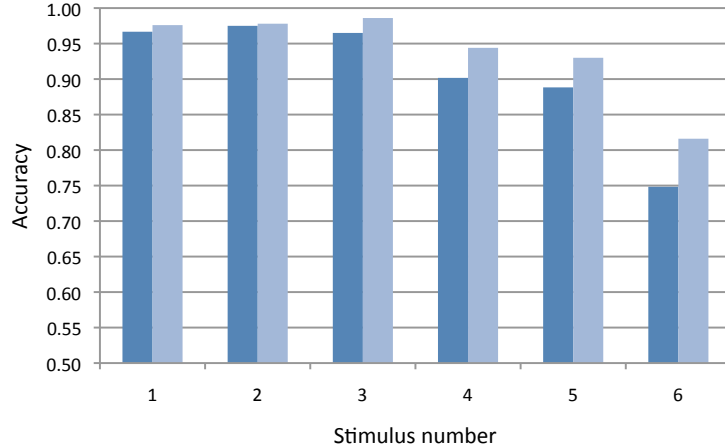


FIGURE 4.8. The proportion of runs for which the model gave the correct output for each of the six stimuli in the 1-3 and 4-6 matching version of the Store/Recognize task. The dark bars are for runs using a decay in the controller’s instruction sequence memory of $k_D = 0$, while the light bars used $k_D = -0.05$. (Note that the vertical axis does not begin at 0.0.)

equivalent to the gain terms that have been used to produce a primacy effect in models of serial recall previously (e.g., Choo and Eliasmith, 2010).

The capacity of the working memory layer was also tested by performing a varying number of load operations, from one to eight, followed by a pair of evaluations. The first evaluation queried the model to see if it recalled being presented with the first stimulus, and the second evaluated a novel stimulus. This should produce an output of *present* followed by *not present*. For example, to test the model’s ability to remember four items, the visual stimuli could have been ABCDAZ (Table 4.3). For these experiments there was no decay in working memory ($k_d = 0$ in Equation 4.1). Figure 4.9 shows the accuracy of the model on the former evaluation. The model was also very successful at correctly identifying the final input as novel; accuracy on that question ranged from 93.2% to 96.5% and was independent of the number of stimuli loaded. Accuracy begins to drop off at a capacity of four, with performance dropping

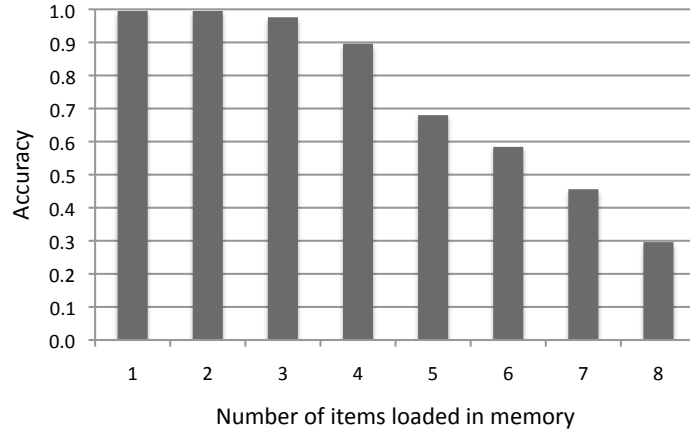


FIGURE 4.9. Results of the capacity tests. After loading one through eight stimuli into working memory, the first item loaded was evaluated. Shown is the proportion of runs which correctly identified that stimuli as having been previously seen.

TABLE 4.3. Example inputs and outputs for a run of the capacity experiments.

Stimulus #	1	2	3	4	5	6
Visual Input	A	B	C	D	A	Z
Mode Input	load	load	load	load	evaluate	evaluate
Correct response	complete	complete	complete	complete	present	not present

under 50% when seven items have been loaded. This corresponds well with measures of human working memory, which have found four items to be a typical capacity level Cowan (2001).

As mentioned, the above capacity experiments do not use any decay in the working memory. Biological working memory tends to exhibit a “recency effect,” in which more recent items are more likely to be successfully recalled than older ones. In order to model this, I have run further capacity tests with a decaying working memory. The results can be seen in Figure 4.10, which shows the degree to which GALIS is able to recall each of six stored inputs for varying values of k_d . When $k_d = 0.0$ the model does equally well recognizing the first of

the six stimuli it has seen as it does the last. In the absence of decay this is to be expected, since the earlier stimuli interfere with the latter exactly as much as the latter interfere with the former. When decay is introduced ($k_d = 0.05$), older memories are degraded as each new input is trained. As a result a recency effect emerges, and GALIS is better able to recall more recent stimuli. As k_d is increased ($k_d = 0.10$) further the model is able to recall the more recent stimulus very reliably, but at the cost of having mostly forgotten about the first stimulus. This relationship between interference and decay is consistent with our earlier studies of working memory described in Chapter 3 (see also Reggia et al., 2009; Sylvester et al., 2010). The memory capacity of this architecture is maximized when there is a trade-off between interference and decay: too much decay and items deteriorate too soon; too little decay and the older items remain in memory interfering with newer ones.

4.3 THE n -BACK TASK

The prior section described a proof-of-concept GALIS model to demonstrate that cognitive control is possible using independent recurrent networks, sequential attractors, and gated connections. This section continues by presenting a second GALIS model capable of performing a more difficult, real-world working memory learning task that is widely used in cognitive psychology. Some lessons learned during the development of the previously presented model were incorporated when constructing this one, resulting in some small changes to GALIS.

In an n -Back task, the participant is presented with a stream of stimuli and must identify which of these is the same as the stimulus presented n steps earlier.

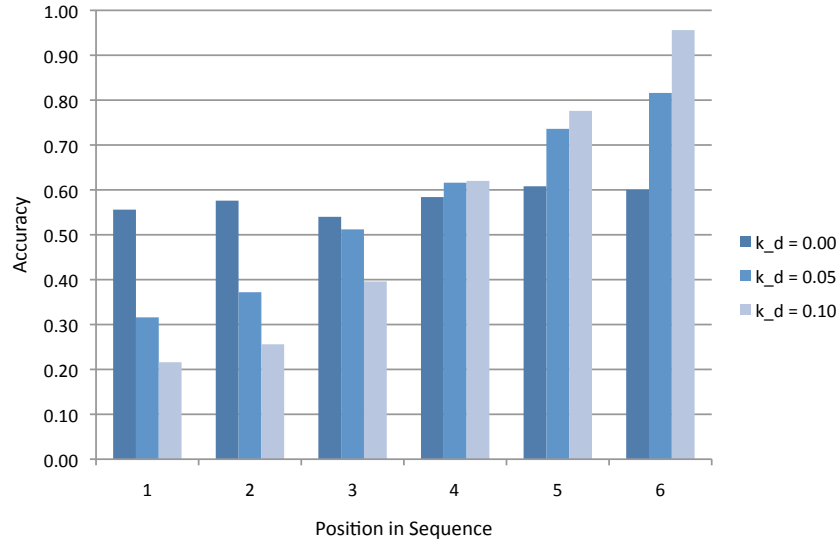


FIGURE 4.10. The effects of decay in the working memory. Without decay the model is equally able to recall any of the six stimuli it has been trained on, without regard to the order in which they were trained. When working memory decay is introduced the model gains the ability to recall more recent stimuli at the cost of making recalling older stimuli less likely.

For example, in a 3-back task the bold letters in the following sequence would be considered matches: **VHZ****V****X****O****L****I****O****S****A****J****X****A****O**. Following each letter (except the first n), the participant must give either a *match* or *no-match* response. The n -Back task is of significant interest in cognitive psychology (Owen et al., 2005). It is commonly used in brain imaging studies (e.g., Schmidt et al., 2009; Watter et al., 2001), correlated with general intelligence (Jaeggi et al., 2008), and used for training to improve working memory capacity (Jaeggi et al., 2010, 2011; *but see* Sprenger et al., 2013, Thompson et al., 2013).

In order to best explain the GALIS model for the n -Back task, which is more complex than that of Section 4.2, the model’s description has been split up into two sections. Section 4.3.1 gives an overview of each component of the GALIS model for n -Back, then describes how it operates to process stimuli, and finally

covers the internal structure of the control module. The intent is to present an intuitive description that makes evident how the GALIS model for n -Back is similar to that for Store/Recognize, and to also highlight the variations due to the different tasks. Having provided a description of the function of the model’s components, further technical details are given in Section 4.3.2.

4.3.1 METHODS

4.3.1.1 TOP LEVEL ARCHITECTURE FOR n -BACK TASKS

The GALIS model for performing n -Back tasks consists of several interacting regions, as seen in Figure 4.11. They are the visual input layer, the n -input layer, the output nodes, the memory layer, the compare module, the context module, and the control module. While there are some differences, overall the functionality is similar to the Store/Recognize task GALIS model described in the preceding section, as will be evident in the following description.

Nodes in the visual input layer are set externally to represent the visual stimulus being presented during the current time step. Visual stimuli take the form of 128-bit, randomly selected bipolar patterns. Patterns are generated such that each input has an equal chance of being either 1 or -1. There are no constraints placed on inter-pattern distances. For ease of discussion we refer to visual stimuli as “letters,” as in the example sequence given above.

The n -input layer is a second input layer which is used to specify the current goal of the model. This module is a stand-in for goal-related information that may be represented biologically in the rostral prefrontal cortex (Charron and Koechlin, 2010), and which could be added in a future extension of GALIS. In the case of modeling the n -Back task, this n -input layer encodes which

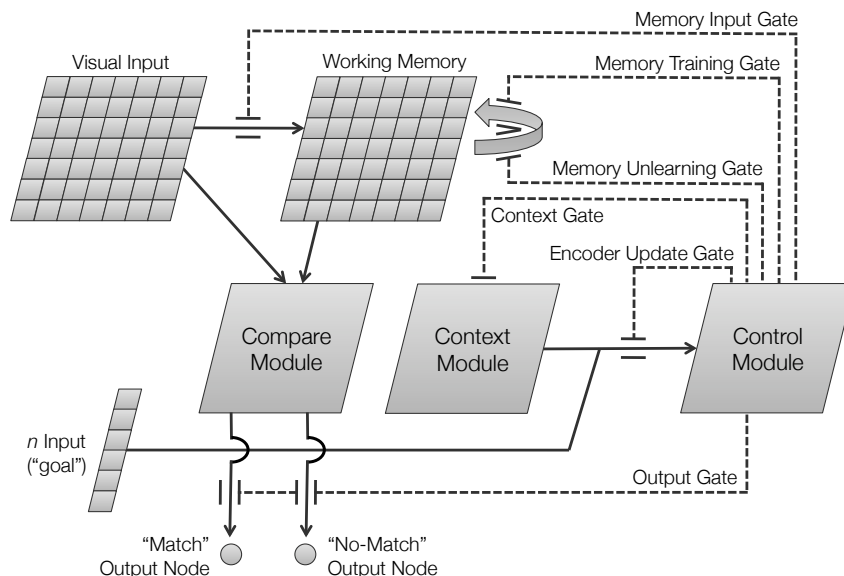


FIGURE 4.11. The GALIS model as used for the n -Back task. Thin, solid arrows denote one-to-one connections. The working memory layer is fully recurrently connected (broad arrow). Dotted lines are the outputs of the control module. Note that the number of boxes pictured in each layer is an approximation only, and does not faithfully represent the number of nodes used in the model.

particular version of n -Back the model should currently be performing (i.e., the current value of n). One of five different distributed patterns is used to indicate whether the model's current objective is to perform 1-, 2-, 3-, 4-, or 5-back. The five specific patterns used are random bipolar patterns chosen in advance. Note that the model learns to execute all five versions. Which version is executed in a specific situation depends only on changing this input, not on retraining or reconfiguring the model in any way. While the input layer used for this example selects only among the relatively limited set of five versions of n -Back, ultimately the same input mechanism could be used to select between a wider array of tasks and objectives.

Two linear threshold units are used for model output, one each for *match* and *no-match*, indicating whether the present stimuli is the same as the one

n steps previously. The inputs to both nodes are gated. They can only be activated when the control module has opened the Output gate.

The working memory layer is a discrete attractor network forming an auto-associative memory, along the lines of those described earlier in this dissertation. Like biological working memories, the GALIS working memory layer has limited capacity (McEliece et al., 1987), high plasticity via one-shot learning (Sandberg et al., 2003), and close integration with executive systems. Additionally, the working memory layer has been modified from standard Hopfield networks to include dynamic thresholds, weight decay and temporally asymmetric weights as outlined in the previous chapter to enable it to recall a temporal sequence of stored patterns in a specified order rather than randomly. The working memory layer is the same size as the visual input layer, and its nodes are bipolar valued. It is treated in more depth in Sections 4.3.1.2 and 4.3.2.1.

As with the Store/Recognize task, the compare module is used to compare the visual input layer to the current state of the working memory, to assess if the current stimuli and recalled stimuli match. Depending on the similarity between the two, it will send activity to one of the two output nodes. Please refer to Section 4.3.2.4 for details.

The context module allows the control module to keep track of what stage of processing it is in. Processing each new stimulus occurs in two stages, called *start* and *finish*. During the *start* phase, the new stimulus is added to the working memory contents. During the *finish* phase the working memory contents is searched to determine if that new stimulus is a match with the n -Back item. The control module adjusts, via the Context Gate, the state of the context module to indicate the current stage. This state information can then be output back to the controller, allowing the controller to affect its own

inputs in the following time step and giving it greater flexibility than if it were to respond only to the current input.¹ In effect, this gives the controller the ability to select its own short-term sub-goals to be carried out in the following time step, similar to the Endogenous Goals layer of CODAM (Korsten et al., 2006). Details are given in Section 4.3.2.5.

The final component is the control module, which is responsible for directing the operation of the rest of the system. It takes input from the n -input layer and context module, and its outputs drive the six gates which govern flow of activity and updating of weights throughout the rest of the model. Just like with the Store/Recognize task, the core of the control module is a second discrete attractor network, called the “instruction sequence memory” (ISM). Like the working memory attractor network, the instruction sequence memory stores sequences using temporally asymmetric weights. But where the working memory module stores visual stimuli, the control module stores the actions necessary for completing a task. Both the working memory and ISM are based on the same weight update, input and state update rules. Reusing the same principles for both data and instruction storage makes GALIS particularly parsimonious. However, the ISM has been modified to store multiple sequences concurrently. Each sequence corresponds to a particular set of actions the model may need to perform during a task. For instance, with n -Back, one such sequence of actions would be used to add a new stimulus to working memory. Each component action takes a single time-step of the simulation to execute,

¹ The context module could be considered as part of the control module, but we indicate it separately here to facilitate explanation. In addition, separating the two increases the modularity of GALIS models by allowing the control module to be agnostic about the source of its inputs.

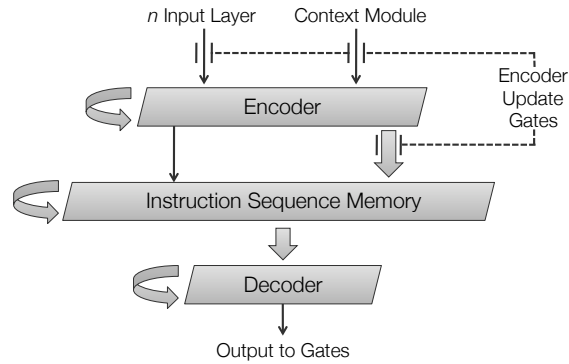


FIGURE 4.12. Slightly expanded control module showing its gated input encoder and output decoder which function as heteroassociative memories. (See Figure 4.14 on page 116 for full detail.)

and corresponds to a particular set of signals to open and close different gates to different degrees.

The control module has two other components besides the ISM, an input “encoder” and output “decoder” (see Figure 4.12). They serve as heteroassociative memories that translate the inputs to the control module into the particular patterns stored in the ISM, and then translate the response of the ISM into the controller’s final outputs. This pre- and post-processing is done primarily to mitigate the effects of noise and crosstalk and to enable the control module to convert between inputs, stored patterns, and outputs of differing dimensions. Details on the control module can be found in Sections 4.3.2.2 and 4.3.2.3.

Control Outputs and Gating. The control module again manages the behavior of the model through a system of gates. The outputs of the control module are used to open and close these gates, which in turn modulate the flow of activity between layers and regulate the weight updates in the working memory layer.

The six gates which control the flow of activity throughout the n -Back model can be seen in Figure 4.11. They are:

1. the Memory Input gate, between the visual input layer and working memory layer that biases the memory's current state towards or away from the current stimulus;
2. the Output gate which controls the flow of activity from the compare layer to the output nodes;
3. the Memory Training gate which controls when the working memory learns a new pattern;
4. the Memory Unlearning gate which controls when the working memory removes a pattern from memory;
5. the Context gate which regulates the state of the context module; and
6. the Encoder Update gate which governs the inputs to the control module, so that it can decide whether it updates its own state.

When a gate is open it allows information to flow through it like an open valve in a water pipe (in contrast to an open switch in an electrical circuit, which prevents flow). A gate's state is given by

$$g^t = k_g g^{t-1} + s^t \tag{4.9}$$

where k_g is a decay term, here equal to 0.5, t is the current time step, and s^t is the current value of the control module output governing this gate.²

² Here t and $t-1$ denote the current and previous time step in the simulation, respectively. This is true of all equations except Eqs. 4.17 & 4.18, in which τ and $\tau-1$ are used in their place to refer to the training epoch rather than the time step. This is by necessity, since the control module learning which these equations describe occurs before the simulation itself is run, i.e., before there are time steps as such to be counted.

The effect of gates is multiplicative, such that the downstream activity of a gated connection is a product of its incoming activity and its current state g . Gates have values in $[-2, 2]$, so gates have more nuanced effects than binary states of “open” and “closed.” A gate can have an amplifying effect on its incoming value ($g > 1.0$), a damping effect ($0.0 \leq g < 1.0$), or an inhibitory effect ($g < 0.0$). Being able to use the same system for both attending to an input (i.e., amplification) and inhibiting that input is appealing, since the two effects can be viewed as antipodal (Engle et al., 1995). An exception to this continuous behavior are the two gates which control learning and unlearning in the working memory. Because updating the working memory weights is a discrete decision — a weight matrix is either updated or not in any time step — these gates have a threshold. Their state is calculated the same way, and the weights are updated when $g > 1$ and not updated when $g \leq 1$. All gates are initially closed when the model begins.

In some situations there are many connections being mediated by the same gate. For instance, the connections between the input and memory layer are one-to-one. These may be thought of as 128 individual connections each having their own gate, with each gate having an identical value. The effect is the same as a single “master” gate controlling all 128 connections based on a single output from the control module, and so we adopt the convention of referring to the parallel opening and closing of these 128 connections as if there is a single gate present.

4.3.1.2 WORKING MEMORY

The working memory layer is a discrete attractor network incorporating dynamic thresholds and temporally asymmetric weights that permit it to process

temporal sequences as described in Chapter 3 as well as Reggia et al. (2009) and Sylvester et al. (2010). The dynamic threshold keeps the network from becoming stuck in any single attractor basin during recall by incrementally increasing the amount of input a node must have in order to stay in the same state. This allows multiple patterns to be activated serially during recall rather than having the network settle on a single pattern, as is typically the case with the fixed-point attractor dynamics of standard Hopfield networks.

The temporally asymmetric weights are formed using a one-shot Hebbian learning rule which correlates a node’s activity with the activity of the other nodes during the presentation of the *previous* input rather than the current one, unlike with typical Hopfield networks. By using correlations between both concurrent and consecutive activity there is the potential for representing more structured information (Cowan, 1999). In particular, the asymmetric weights are used here to ensure that the network not only switches between attractors in its state space, but does so in an order corresponding to that in which the input patterns were presented. Section 4.3.2.1 provides details on weight learning rules, calculating inputs and updating states.

In addition to adding patterns to working memory, the network also has the capability to “unlearn” or partially “forget” stored patterns. This is accomplished using an anti-Hebbian learning rule (Hopfield et al., 1983). One could think of the unlearning procedure as the addition of an “erase” command to complement the typical “load” and “store” functions already present. In the case of n -Back, for example, patterns more than n steps back in the sequence are no longer needed. Unlearning these patterns reduces the interference they cause, making it easier to recall more recent stimuli. While this model was able to perform n -Back tasks without needing to unlearn these older stimuli, initial

experiments indicated that unlearning significantly increased performance due to reduced interference.

4.3.1.3 MODEL OPERATION

Each run of the model is divided into two phases: Controller Initialization and Task Execution. In the Controller Initialization phase, the control module learns the instruction sequences necessary to perform the task using one-shot Hebbian learning. This is so the ISM contains the appropriate pairings of conditions and responses when the task is begun. This training of the control module occurs only once, and after the Task Execution phase begins its weights remain unchanged. The working memory layer, in contrast, begins in a blank, untrained state, and has its weights updated multiple times as the trial progresses through the Task Execution phase. While the associations being learned in the control module are determined by the human modeler, the learning that the working memory engages in during the task is entirely guided by the model itself, with the model determining when to add or remove a pattern from working memory.

During each step of processing in the Task Execution phase the model goes through the following operations, directed by the control module. If the model activated either output node in the previous time step, a new stimulus will be presented, otherwise the inputs from the previous step are retained. Next, the state of the working memory is updated. Then the output of the compare module is updated to reflect the new state of working memory and the potentially new state of the visual input layer. Following this the state of the context module is updated.

Next, the control module’s encoder is updated if the Encoder Update gate, which regulates it, is open. If it is not open, the encoder input will be the same as the previous time step, and thus the downstream layers in the control module will receive the same inputs as the previous time step. This prevents the control module from starting recall of a new sequence of actions before the previous sequence has concluded. (Each action is one of the elements in the sequences stored in the control module, and corresponds to one particular operation necessary to carry out the task, as explained below.) This occurs because the Encoder Update gate is opened only at the end of a sequence, setting the stage for the next sequence to begin at the following time step. The encoder’s output, whether it is the result of new inputs from an open gate or not, is then used as input for the ISM. Once the ISM is updated the decoder selects an action and outputs the gate control signals which compose that action. These newly produced gate control signals are used to update the gating values according to Equation 4.9. If either the learning or unlearning gates are open, then weight updates of the working memory layer occur. If the Output gate is open then either output node may be activated, depending on the state of the compare module. If either is activated then a new stimulus will be presented in the following time step, proximately corresponding to a self-paced stimulus presentation.

4.3.1.4 CONTROLLER FUNCTIONALITY

As with the Store/Recognize task, the Controller Initialization phase occurs before the model is presented with any inputs or produces any outputs. For the n -Back model, the network learns to perform the task for $n \in \{1, 2, 3, 4, 5\}$. The model is always trained to do all five versions, so training is identical

no matter which versions the model will ultimately perform during the Task Execution phase, when the n -input layer specifies which of the five different versions the model will perform. This makes the model capable of switching between versions of n -Back during trials, as dictated solely by its inputs and without any other adjustments being made.

For the n -Back task, during the Controller Initialization phase the control module learns six instruction sequences (Table 4.4; WM = working memory). One of these six sequences adds a new stimulus to working memory when it is executed. The other five each correspond to one of the five possible values of n . Each of these five sequences steps back through the working memory's record of the recent visual stimuli the appropriate number of items and then evaluates whether the current stimulus matches the one recalled. Which of the six instruction sequences is executed is determined by the control module's inputs, which come from the context module and the n -input layer. (See Section 4.3.2.2 for further information.)

To illustrate how the trained control module works during the Task Execution phase on a concrete sequence of inputs, consider a sequence of stimuli $A S D F G$, with A being the first stimulus and G the last. Figure 4.13 illustrates the step-by-step actions that occur in processing the single input pattern G , where the n -input value is 3. The goal is for the model to generate the correct *no match* output since G does not match the 3-back stimulus S . In order to evaluate if G matches the 3-back stimulus (which in this case is S), The model must first add G to its working memory and then recover the 3-back stimulus from its record. This requires stepping backwards through the sequence it has learned, from G to F to D and finally to S .

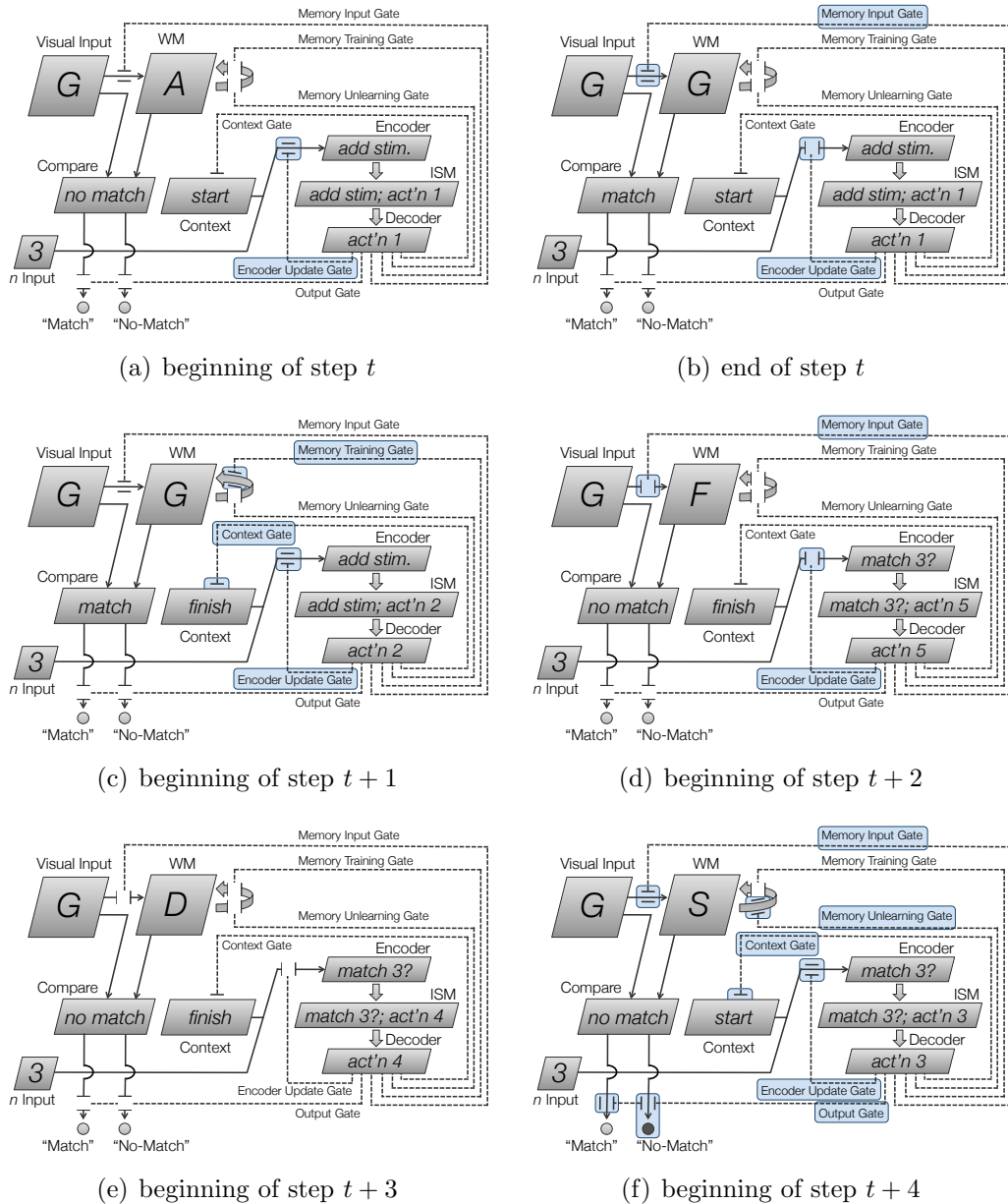


FIGURE 4.13. Step-by-step operation of the n -Back model to process one input stimulus G . (See text for details.)

TABLE 4.4. Instruction sequences learned by the control module’s ISM.

Sequence	Action
1. add stimulus to memory	1. open memory input gate, strongly biasing memory state towards input
	2. train the working memory layer; switch to <i>finish</i> context
2. current stimulus matches 1-back?	3. open output gate; switch to <i>start</i> context; unlearn WM
3. current stimulus matches 2-back?	4. delay (i.e., update the state of WM, but nothing else)
	3. open output gate; switch to <i>start</i> context; unlearn WM
4. current stimulus matches 3-back?	5. delay
	4. delay
	3. open output gate; switch to <i>start</i> context; unlearn WM
5. current stimulus matches 4-back?	6. delay
	5. delay
	4. delay
	3. open output gate; switch to <i>start</i> context; unlearn WM
6. current stimulus matches 5-back?	7. delay
	6. delay
	5. delay
	4. delay
	3. open output gate; switch to <i>start</i> context; unlearn WM

In Figure 4.13(a) a new stimulus **G** is presented for time step t , n -input is set to 3, and the Encoder Update gate is open as indicated by the shaded label in the illustration. Recall that an open gate allows the flow of activation in the same way that an open valve in a pipe allows the flow of fluid. Because $n=3$, the controller determines that it must execute the first instruction sequence in Table 4.4 (“add stimulus to working memory”). The working memory state is depicted as **A** because that is three stimuli before the stimulus which was just processed (**F**). By the end of time step t (Figure 4.13(b)), the effects of action 1 can be seen: the Encoder Update gate has been closed to allow

sequence 1 to finish executing, and the working memory state has become the same as the visual input because the Memory Input gate is open. At the end of $t + 1$ (Figure 4.13(c)), the Memory Training gate has been opened, updating the working memory weights, the Context gate has been closed, changing the context to *finish*, and the Encoder Update gate has been opened to allow a new sequence to be selected in the next time step. At the end of $t + 2$ (Figure 4.13(d)), a new instruction sequence has been selected (“does the current stimulus match the one from three back?”). The Memory Input gate is closed, allowing the working memory to recall the previous item in memory. The Encoder Update gate is closed again to allow the instruction sequence to complete. In Figure 4.13(e) the gates remain unchanged as the working memory recalls the preceding item again. In Figure 4.13(f) the working memory steps back a third item in memory. The Output gates are opened, allowing the compare module to activate the *no-match* node since the input pattern G fails to match the working memory state S , generating the correct output for this stimulus. The Memory Unlearning gate is opened to forget S now that it is no longer relevant to the task. The Context, Memory Input and Encoder Update gates are switched to ready the model for a new stimulus in the following time step. Other values of n would lead to similar behavior, only with a different number of delaying steps until the match step in Figure 4.13(f) is done.

4.3.2 MODEL DETAILS

This section provides some further details on the n -Back model that are relevant for its operation but not necessary in order to understand its overall structure. Two systems are particularly highlighted here as they differ most from the

model of the prior section. The first is the working memory, which differs because it must process sequential memories to perform n -Back, which it did not need to do to for the Store/Recognize task. Second is the controller. The controller architecture used for Store/Recognize is slightly simplified for ease of explication, whereas for n -Back it is more refined. Further details on the controller are provided here since this is the form that will be used in the final two chapters.

4.3.2.1 WORKING MEMORY

The working memory layer is based on the temporally asymmetric attractor approach developed in Chapter 3. It uses two weight matrices to store items in the sequence as well as their order. The first weight matrix, W_{WM} , is trained with standard one-shot Hebbian learning with the addition of a weight decay term so that older memories are supplanted by more recent ones:

$$w_{ij}^t = (1 - k_{\text{WM}}) w_{ij}^{t-1} + \frac{1}{N} a_i^t a_j^t (1 - \delta_{ij}) \quad (4.10)$$

Here k_{WM} is the decay rate ($0 \leq k_{\text{WM}} < 1$) and δ_{ij} is Kronecker's delta, which ensures that weights on self-connections are fixed at zero. The second weight matrix, V_{WM} , also uses Hebbian learning but associates the state of a node not with the current states of other nodes, but with the other nodes' *previous* states. This introduces a sense of temporal ordering to V_{WM} , making it possible to recall the stimuli in order rather than randomly. The learning rule is given by

$$v_{ij}^t = (1 - k_{\text{WM}}) v_{ij}^{t-1} + \frac{1}{N} a_i^t a_j^{t-1} \quad (4.11)$$

The decay term is still present, although the Kronecker’s delta factor is not as it is desirable for a node’s activity to be influenced by its own previous state. Updating the state of the working memory layer occurs in two stages, the first governed by the asymmetric weights, and the second by the symmetric weights. When both W_{WM} and V_{WM} are used simultaneously they can work at cross purposes. V_{WM} is pushing the network towards the next attractor, while W_{WM} is fighting to keep it in the same attractor basin. The two-stage process adopted here helps the network to proceed from one state to the next in a more orderly and predictable progression.

Activation updating begins by first calculating the input h_i^t to each node i using V_{WM} and the previous network state along with a gated connection from the topologically corresponding node in the input layer. Using only V_{WM} to update the network serves to move the network state from the current attractor basin to the basin associated with the next pattern in the sequence:

$$h_i^t = \sum_j v_{ij} a_j^{t-1} - \theta_i^t + 2 g_{in}^t in_i^t \quad (4.12)$$

where v_{ij} is the strength of the temporally asymmetric connection from node j to node i both in the memory layer, a_j is the state of node j in the memory layer, g_{in} is the value of the gating node mediating the input-to-memory connections, in_i is the state of node i in the visual input layer, and θ_i is a dynamic threshold that is used to keep the network from settling permanently into any one attractor basin. If a node’s state has not changed in the previous time step, the magnitude of θ_i increases, which means node i will require inputs with larger magnitudes to remain in the same state. This is done according to the same procedure as for Store/Recognize (see § 4.2.4.1 as well as Equation 3.4), and the same parameter values are used — namely $k_\theta = .02$ and $k_w = 0.0125$.

The input h_i is then used to update the state of each node according to the same step function used previously (i.e., Equations 3.5 and 4.3).

After updating both \vec{a} and $\vec{\theta}$, the updating process begins again, this time using W_{WM} and the current network state to calculate the input f_i^t according to the following rule:

$$f_i^t = \sum_j w_{ij} a_j^t - \theta_i^t + 2 g_{\text{in}}^t \text{in}_i^t \quad (4.13)$$

This helps the network to settle further into the new attractor basin it was pushed towards by V_{WM} in the previous stage. The asymmetric weights suffice to get the network into the next attractor basin; the symmetric weights impel it into the bottom of that basin, reducing the noisiness of the recall. This new input f_i^t is then used to update \vec{a} according to Equation ?? again (though the conditionals are predicated on f_i , not h_i , this time), and $\vec{\theta}$ is updated once again.

The working memory module's unlearning is defined by the following anti-Hebbian rules:

$$w_{ij}^t = w_{ij}^{t-1} - \frac{1}{2^{n-1}} \cdot \frac{1}{N} a_i^t a_j^t (1 - \delta_{ij}) \quad (4.14)$$

$$v_{ij}^t = v_{ij}^{t-1} - \frac{1}{2^{n-1}} \cdot \frac{1}{N} a_i^t \text{sgn} \left(\sum_{l=1}^N v_{lj} a_l^t \right) \quad (4.15)$$

Here N is the number of nodes, while n is the same as the lag n in n -Back. The a_j^{t-1} term in Equation 4.11 has been replaced in Equation 4.15 by the summation because the goal is not to disassociate the current state of the memory from the state immediately preceding it, but from the pattern which was *trained* preceding the current one. Because V_{WM} is trained to make the memory move toward the previously trained pattern, we can use it to approximate the pattern

trained prior to the current state. The factor of $1/2^{n-1}$ is used so that unlearning is more aggressive when shorter sequences need to be retained.

4.3.2.2 CONTROLLER OPERATION

In the Task Execution phase, processing each visual stimulus occurs in two stages. The objective of the first stage is to add the new stimulus to working memory. This situation is indicated by the context module outputting the *start* pattern. If the context module is outputting *start*, then sequence 1 in Table 4.4 will be executed, regardless of the value of the n -input layer. Adding a new item to working memory is accomplished through two actions (numbers 1 & 2 in the table). The first action opens the Memory Input gate, so that the state of the working memory will be biased towards the current visual stimulus. The second action updates the weights W_{WM} and V_{WM} to add the current state to working memory, and switches the context module to the *finish* mode so that when the control module updates in the next time step it knows that the working memory has already been trained.

The objective of the second stage in processing a visual stimulus is to evaluate whether that stimulus matches the one presented n steps ago. This requires stepping back through the working memory's record of events, which is accomplished by allowing the working memory's dynamics to run n times. Due to the effect of the temporally asymmetric weights, the state of the network should shift to the previously trained item each time it updates. The sequence to carry this out is selected based on the value of the n -input layer as well as the context module outputting the *finish* pattern. For clarity, we describe how this works in detail only for 3-back since extrapolation to the other versions of the task is straightforward.

Checking whether the current stimulus matches the stimulus three items ago requires three actions (numbers 5, 4 & 3 in Table 4.4) executed consecutively. The first two actions each delay the controller for a time step, which gives the working memory the opportunity to update its state twice. During each of these updates the asymmetric weights should move the network to the previously trained stimuli. The third action does three things: open the Output gate so the comparison between the current memory state and visual input can be output, unlearn the current state of the working memory since it is no longer relevant to the task, and switch the context module back to *start* so that in the next time step the controller will know that processing the current stimulus is complete and it is time to begin the first phase of processing the next stimulus.

(There are multiple delay actions with identical effects listed in Table 4.4 because the ISM cannot be trained to repeat the same pattern a set number of times, such as a sequence like $\alpha, \alpha, \alpha, \delta$. It can, however, learn $\alpha, \beta, \gamma, \delta$. By defining β and γ to cause the same controller outputs as α —that is, if you define them to be three different tokens all of the same type—you can reproduce the effects of training the sequence $\alpha, \alpha, \alpha, \delta$. The four different actions labeled “delay” all have the same effect, but each is represented as a distinct bipolar pattern. Using this type/token distinction, the number of total time steps the working memory delays can be controlled by using a different number of these delay tokens in the instruction sequence for each value of n .)

4.3.2.3 CONTROLLER ARCHITECTURE

A diagram of the control module’s internal structure can be seen in Figure 4.14. It is composed of three subcomponents. The principal of these is the instruction sequence memory (ISM) which learns the actions needed to respond to each

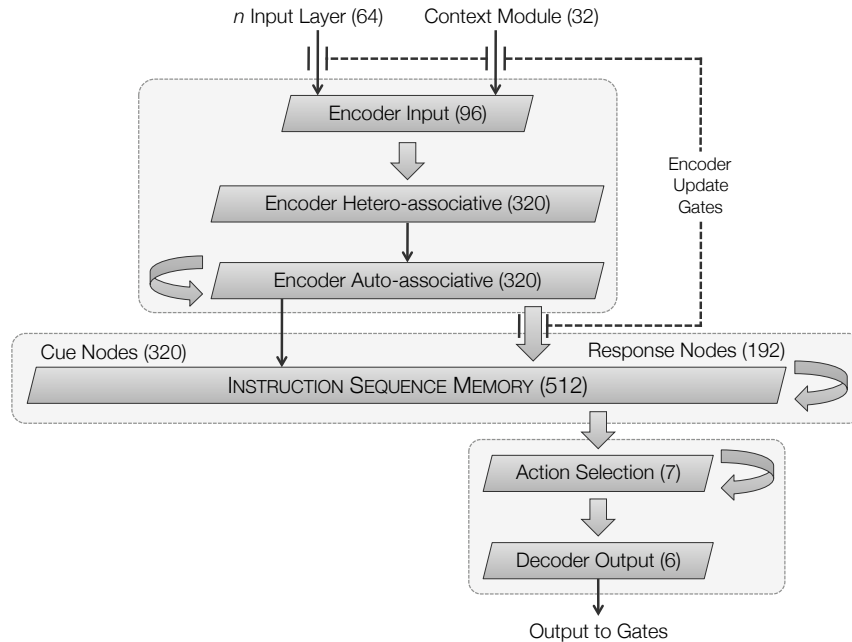


FIGURE 4.14. The control module. Thick arrows denote fully connected layers, while thin arrows denote one-to-one connections. Following the name of each layer is the number of nodes it contains in the example presented in this paper. One-to-one connections from the Encoder Auto-associative layer terminate on the cue nodes of the instruction sequence memory, while full connections terminate on the response nodes.

circumstance. The other two components are the encoder and decoder, which are used for pre- and post-processing to convert the inputs of the controller into the patterns stored in the instruction sequence memory, and then from those patterns into the gating control signals the control module outputs.

Instruction Sequence Memory. The instruction sequence memory is a discrete autoassociative memory that uses temporally asymmetric learning in addition to standard Hebbian learning to process sequences (Chapter 3, *supra*; also Sylvester et al., 2010). This allows it to store which actions make up the response needed for the task, and also the order in which those actions must be carried out. The ISM has one important difference from the working memory

layer, however: it has been modified to store multiple sequences at the same time. This is accomplished by way of a conceptual division of the nodes into two sets, the “cue” and “response” nodes. (The working memory module could also be augmented in this way, for instance to model a dual n -Back task (Jaeggi et al., 2010), but it is not necessary for this model.)

The role of the cue nodes is to provide the necessary context information to the network to select from among the stored instruction sequences. The state of the cue nodes corresponds to the situation the model is facing. The response nodes are responsible for storing the actual items in each sequence, and thus selecting an action from those in the given instruction sequence. Each instruction sequence and each action are represented internally by random bipolar strings. For n -Back there are six different instruction sequences and seven actions, outlined in Table 4.4. Because an action can belong to more than one sequence there are a total of seventeen patterns stored as attractors in the ISM (one for each row of Table 4.4).

Although the ISM is divided into cue and response groups, its nodes are fully connected. The difference between the two types of nodes lies in their inputs and outputs. Only the state of the response nodes are output to the decoder and the cue and response nodes receive different inputs from the encoder. Cue node i 's external input e_i comes from one-to-one topographic connections from the corresponding node in the encoder auto-associative memory. These connections allow the cue pattern which has been chosen by the encoder to be passed on to the cue nodes. Response nodes, on the other hand, are fully connected to all nodes in the encoder auto-associative memory. The weights on these connections are trained using one-shot Hebbian learning to associate each cue pattern with the first response pattern in that sequence. The purpose

of the connections between the encoder and response nodes is to bias the ISM towards the first pattern in the sequence. This is only desirable when a new sequence is being selected, so the gate on these connections is kept closed at all other times. This way the encoder influences the response nodes only in time steps when the controller determines that a new sequence is supposed to be selected, and is ignored otherwise.

The external input to ISM node i is defined by

$$e_i = \begin{cases} a_{\text{enc}_i} & i \in \{\text{CUE}\} \\ g_{\text{ctrl}} \sum_{k \in \text{ENC}} u_{ik} a_{\text{enc}_k} & i \in \{\text{RESPONSE}\} \end{cases} \quad (4.16)$$

where a_{enc_j} is the state of node j in the encoder auto-associative memory, u_{ik} is the connection strength from node k in the encoder auto-associative memory to node i in the instruction sequence memory, ENC is the set of nodes in the encoder, and g_{ctrl} is the value of the Encoder Update gate.

Like the working memory, the ISM also has two weight matrices, W_{ISM} and V_{ISM} . The former is trained using standard Hebbian learning and the latter using temporally asymmetric learning, defined by the following rules:

$$w_{ij}^\tau = (1 - k_{\text{ISM}}) w_{ij}^{\tau-1} + \frac{1}{N} a_i^\tau a_j^\tau (1 - \delta_{ij}) \quad (4.17)$$

$$v_{ij}^\tau = (1 - k_{\text{ISM}}) v_{ij}^{\tau-1} + \frac{1}{N} a_i^{\tau-1} a_j^\tau \quad (4.18)$$

where τ is the training epoch. Here \vec{a} is simply the concatenation of a cue and response pattern which make up one of the seventeen distinct actions listed in Table 4.4. Unlike k_{WM} , k_{ISM} can be positive or negative. When negative, it acts as a gain rather than decay. A positive value decreases the strength of earlier items in a sequence. This is desirable when trying to reproduce

serial position effects in human memories of external stimuli, but there is no particular reason for earlier items in the instruction sequence to be diminished. Negative values, which serve to amplify earlier items, have surprisingly been found to be beneficial for the ISM.

Other than taking input from the encoder rather than the visual input layer, the dynamics of the ISM are the same as those of the working memory. The same two-part update process as the working memory layer (using first the asymmetric and then the symmetric weights), although Eqs. 4.12 and 4.13 are redefined as follows to accommodate the differences in the external input, given in Equation 4.16.

$$h_i^t = \sum_j v_{ij} a_j^{t-1} - \theta_i^t + e_i^t \quad (4.19)$$

$$f_i^t = \sum_j w_{ij} a_j^t - \theta_i^t + e_i^t \quad (4.20)$$

Both the state \vec{a} and dynamic threshold $\vec{\theta}$ of the ISM are updated in the same way as they are in the working memory.

Encoder. The encoder is responsible for selecting an instruction sequence to execute by translating between the inputs to the control module and the cue portion of the patterns stored in the ISM. Both the inputs to the encoder and the connections between the encoder and the ISM response nodes are gated. They are opened as part of the final action of a sequence, which allows the control module to ready itself in the next time step to begin processing a new sequence of actions. When the Encoder Input gate is open activity flows from the context module and n -input layer to the encoder's input layer. When this gate is closed the encoder does not receive input so the state of its input layer

remains unchanged from the previous time step. This in turn means that the encoder's output will be the same as the previous time step, so the ISM will be operating on the same cue pattern as it did in the previous time step. This is done to prevent the ISM from switching to the next instruction sequence before the previous sequence is completed. The encoder-to-response-node connections are mediated by the same gate value, since they should influence computation when a new sequence is being started and be ignored otherwise.

The encoder is composed of three layers of bipolar nodes (Fig. 4.14). This architecture could possibly be made simpler, but only at the expense of more complicated dynamics. There are 96 nodes in the first layer (one per input) and they receive input from the output of the context module and the n -input layer. The second and third layers have 320 nodes, one per cue node in the ISM. The connections between the first and second layers are trained by one-shot Hebbian learning, forming the two layers into a heteroassociative memory which can produce the correct cue pattern when given the corresponding n -input vector and context module output.

The third layer is a standard autoassociative Hopfield network which outputs to the ISM. It has a full set of intralayer connections which have been trained by one-shot Hebbian learning to recognize cue patterns. These connections serve to move the state further into the current activity basin, i.e., closer to the cue pattern that the heteroassociative memory recalled. This mitigates errors resulting from the first two layers. By training only on the cue patterns and not the control module's inputs or response patterns, this layer specializes in cue pattern memory, and avoids interference which can arise in the preceding and following layers.

This autoassociative memory is carrying out a “redintegration” process, using prior knowledge to help reconstruct a pattern from a partial copy (Baddeley, 2007). Many cognitive models require that a pattern retrieved from memory be reconstructed in some way (Lewandowsky and Farrell, 2003), including CLARION (Sun, 2006) and the “clean-up” memory of Eliasmith and colleagues (presented in (Stewart et al., 2011) and used in (Choo and Eliasmith, 2010), among others). Attractor networks have been used for this purpose before, including Lewandowsky (1999) and Kesner et al. (2000).

Decoder. The decoder is responsible for translating the patterns represented in the ISM response nodes into the signals used to drive gate activity. This is accomplished through a competitive layer which serves to select a single response action and a set of Hebbian-trained weights which learn to produce the desired gate outputs for each action. It is composed of two layers (Fig. 4.14). The first, called the action selection layer, has seven binary nodes — as many as there are response patterns.³ It is fully connected to the response nodes of the ISM. These connections are trained by one-shot Hebbian learning to associate each response pattern with a single active node in the action selection layer.

In order to help ensure that only one node will be active, the action selection layer also has a set of recurrent connections which create competitive dynamics, with every node reinforcing its own activation while inhibiting that of the

³ While using one node per action is a violation of GALIS’ commitment to using distributed representations, there is some basis for their use in this situation. Distributed systems using localized nodes for action selection is relatively common (e.g., Amos (2000)). This is partially because it is an effective and convenient arrangement, but also because action selection has been linked to the basal ganglia (Gurney et al., 2001a; Redgrave et al., 1999; Schroll et al., 2012), and the basal ganglia have up to one thousand times as many inputs as outputs. This topology suggests that information is being condensed or integrated in some way, which is what occurs in the decoder. While we are not attempting to explicitly model the basal ganglia here, we still do not wish to ignore the role the cortico-basalganglio-thalamic loops play in action selection.

other nodes. Since the Hebbian weights from the response nodes to the action selection layer have already produced an activity pattern which is close to having a single winner, only a single step of these competitive dynamics is needed to make one node maximally active and all other nodes off.

The action selection layer then feeds in to the control module's output layer, which generates the gate control signals. The weights on these connections also use one-shot Hebbian learning to learn the desired gate control signals. For instance, if action i necessitates fully opening the first gate, partially closing the second, and leaving the others unchanged, action node i 's outgoing weights would be $(1, -0.25, 0, 0, 0, 0, 0)$.

4.3.2.4 COMPARE MODULE

The compare module is composed of two layers. The first is the same size as the memory and visual input layers, and receives one-to-one connections from each of those components. The state of nodes in this layer is the product of the states of the corresponding nodes in the input and memory layers. The second layer has two nodes, both of which take as input a value proportional to the inner product of the input and memory layers' states. One of the second layer nodes adopts a state of one if its input is above a certain threshold—here equal to 0.9—and zero otherwise, while the other node outputs one if its input is below that threshold and zero if it is above. These two nodes drive the model's output nodes.

4.3.2.5 CONTEXT MODULE

Within the context module, two linear threshold units, one each for *start* and *finish*, are each fully connected to a set of 32 nodes. The *start* node activates

when the Context gate’s state is less than one, the *finish* node when it is greater than one. The weights on these connections are randomly selected binary patterns. By raising or lowering the gate’s value above or below the threshold the control module can select one of the two patterns for output. While the context module may seem complex for its relatively simple function, we note that this is largely due to the commitment to using a gating-based mechanism and the desire to maintain modularity between the control and context functions.

4.3.3 RESULTS

After the GALIS n -Back model was trained to perform n -Back tasks of varying lengths ($n=1$ through $n=5$), it was given sequences of $30 + n$ stimuli. The first n are “preparatory stimuli,” and the response to these is ignored. This is done for two reasons: primarily, because this is the way human subjects are evaluated, and secondarily, because the first stimuli present a boundary case to the model for which it was not given special behaviors to handle, namely, attempting to recall a sequence which is longer than the one it has stored. For each trial, ten stimuli would be generated, and the sequence of inputs would then be drawn from these ten. A subset of all possible stimuli was used because trials with human subjects often use limited sets of stimuli such as the digits 0–9 (Schoofs et al., 2008) or eight rotational positions around a circle (Hockey and Geffen, 2004). Of the stimuli following the preparatory period, one third were randomly selected to be matches. The following is a sample sequence used for the 3-back version of the task with matching stimuli emphasized.

ADJAEFDKJCKAHFAHGDFGDKACKHCAGJAGK

A is the first stimulus in the sequence and K is the last. Each sequence was generated without any “lures” (matching stimuli which are one position off from the target location, for instance a match four positions back when doing a 3-back task). Lures were excluded for two reasons. The first is that the human data we were attempting to match (Watter et al. (2001)) did not use lures. The second is that we wished to remove one potentially confounding factor in order to concentrate on investigating the control module’s basic ability to govern the model.

In Figure 4.15, the model’s performance is given, and is also compared to that of human subjects, on 1-, 2- and 3-back tasks. Human data is taken from Watter et al. (2001), which is typical of human results reported in the literature. The model results are the average accuracy across all 30 stimuli in 250 random sequences. The error bars in Figure 4.15 represent the standard error of the mean. Two different variations of the model were tested. Model V used variable working memory decay rates (the larger n was, the smaller the decay rate used, so that for larger n values the working memory attempted to store more stimuli), while Model C used the same decay rate for all values of n .⁴

Both models show that, as n increased from $n=1$ to $n=5$, response accuracy decreased monotonically. For $n=1, 2, 3$ both models’ results are not significantly different than human performance at the level of $p = .05$; however the overall fit for Model V was closer. This is possible because different decay rates are most suitable for recalling sequences of different lengths (Chapter 3). A lower decay

⁴ Specifically, in terms of the parameters given in §4.3.2, Model V used $k_{WM} = .350, .300, .225, .150, .075$ for $n=1$ through $n=5$, respectively, while Model C used $k_{WM} = .2625$ for all versions. These values were chosen via iterative deepening depth-first search. In both models $k_{ISM} = -0.3$ for all versions of the task.

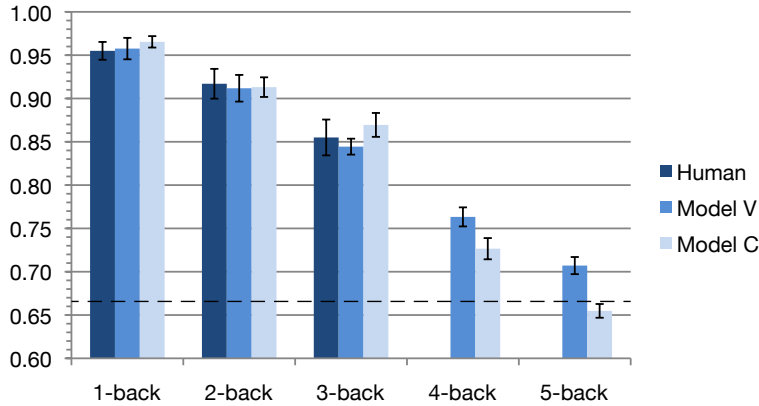


FIGURE 4.15. Accuracy for human subjects and the computational model for all five versions of n -Back. Human results were not reported for $n=4$ and $n=5$. Error bars represent the standard error of the mean. Model V used different working memory decay rates (k_{WM}) for each n , while Model C did not vary decay. The dashed horizontal line indicates the accuracy that would be expected if one randomly selected responses in a proportion matching that used to generate the stimuli — i.e., outputting “match” according to a Bernoulli process with $p=1/3$. (Note that the vertical axis does not begin at 0.0.)

rate in the working memory layer allows longer sequences of visual stimuli to be successfully stored without deteriorating away. A higher decay rate removes older items from memory, reducing interference and improving the ability to recall shorter sequences. This accords with previous investigations into the role of decay on attractor net working memories, where it has been hypothesized that humans may adjust a working memory decay rate in order to control the length of sequences that they are attempting to remember (Altmann and Gray, 2002; Winder et al., 2009). The additional degree of freedom in Model V may account for its improved fit compared to Model C. However, it should be noted that this freedom is not necessary for Model C to produce a statistically significant match with human performance.

Figure 4.15 does not show human results for $n=4$ and $n=5$ because they are not reported in Watter et al. (2001). This is common, as human

subjects typically find them to be extremely challenging (Owen et al., 2005). Nonetheless, the GALIS n -Back model is trained to perform 4- and 5-back, and the simulation results are shown as model predictions. If humans really can adjust working memory decay to adapt to longer sequences, Model V's performance leads us to predict that subjects taking Watter et al.'s version of n -Back for $n=4$ and $n=5$ would see their performance drop off linearly to approximately 76.3% and 70.7%, respectively. Higher values of k_{WM} have more of an impact on larger values of n , since decay is compounded. Keeping $k_{WM} = .2625$ in Model C therefore disproportionately impacts performance for $n=4, 5$, reducing Model C's accuracy on 5-back to no better than chance. (Since one third of stimuli are matches, a strategy of random guessing would result in an expected accuracy of 66.6%.) If humans cannot adjust working memory decay to suit the task then we would predict that their accuracy on Watter et al.'s version of 4-back to fall to 72.7%, and for human subjects to be unable to perform 5-back at better than random accuracy. Both Model V's and C's errors in these more demanding versions of n -Back appear to be caused by the difficulty of recalling sequences of this length from the working memory layer, rather than from improper retrieval of the instruction sequences from the ISM.

The GALIS n -Back model exhibits a response time which is approximately linear in the value of n . When a new stimulus is presented the model requires two time steps to execute actions 1 and 2 in Table 4.4, and n additional time steps for memory retrieval. Watter et al. (2001) also reports the participants' average response times following each stimulus on 1-, 2- and 3-back tasks, which is also roughly linear in n . This is compared to the average number of time steps needed by the GALIS n -Back model in Figure 4.16. The GALIS

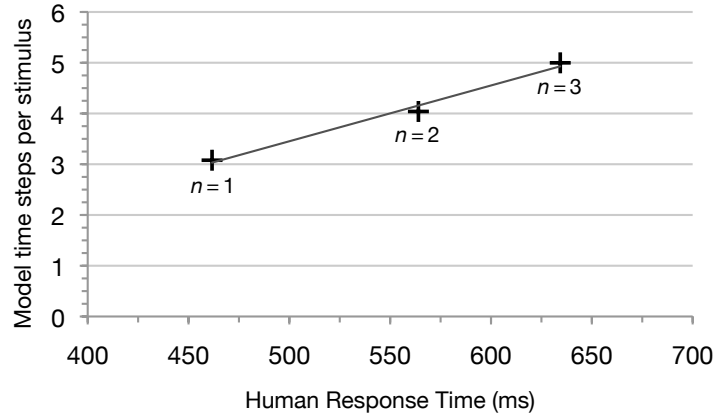


FIGURE 4.16. Correlation between human response time per stimulus and the average number of steps the model takes per stimulus. The trend line is defined by $y = 0.011x - 2.042$, with $R^2 = 0.9888$.

n -Back model's response time correlates well with the human response time data, with $R^2 = 0.9888$. These results are relatively robust to variations in k_{WM} and k_{ISM} .

To demonstrate that our n -Back model is capable of switching between versions of n -Back without relearning any of the instructions, experiments were run in which the value presented to the n -input layer changed mid-trial. Input sequences were constructed in the same way as described at the beginning of this section, with one third matches, no lures, and a preamble of preparatory inputs. For the first fifteen stimuli following the preamble, the n -input layer was given an input of n_1 . Beginning with the sixteenth stimulus, the value of the n -input layer was set to $n_2 \neq n_1$.⁵ No parameters were adjusted or weight matrices were externally modified between the first and second phases of each trial; the only difference was the value of the n -input layer.

Figure 4.17 shows some representative results when mid-trial changes in n occur. Each graph shows the accuracy at each position in the input sequence,

⁵ For all trials in these experiments, $k_{ISM} = -.3$ and $k_{WM} = .225$. (See Section 4.3.2.)

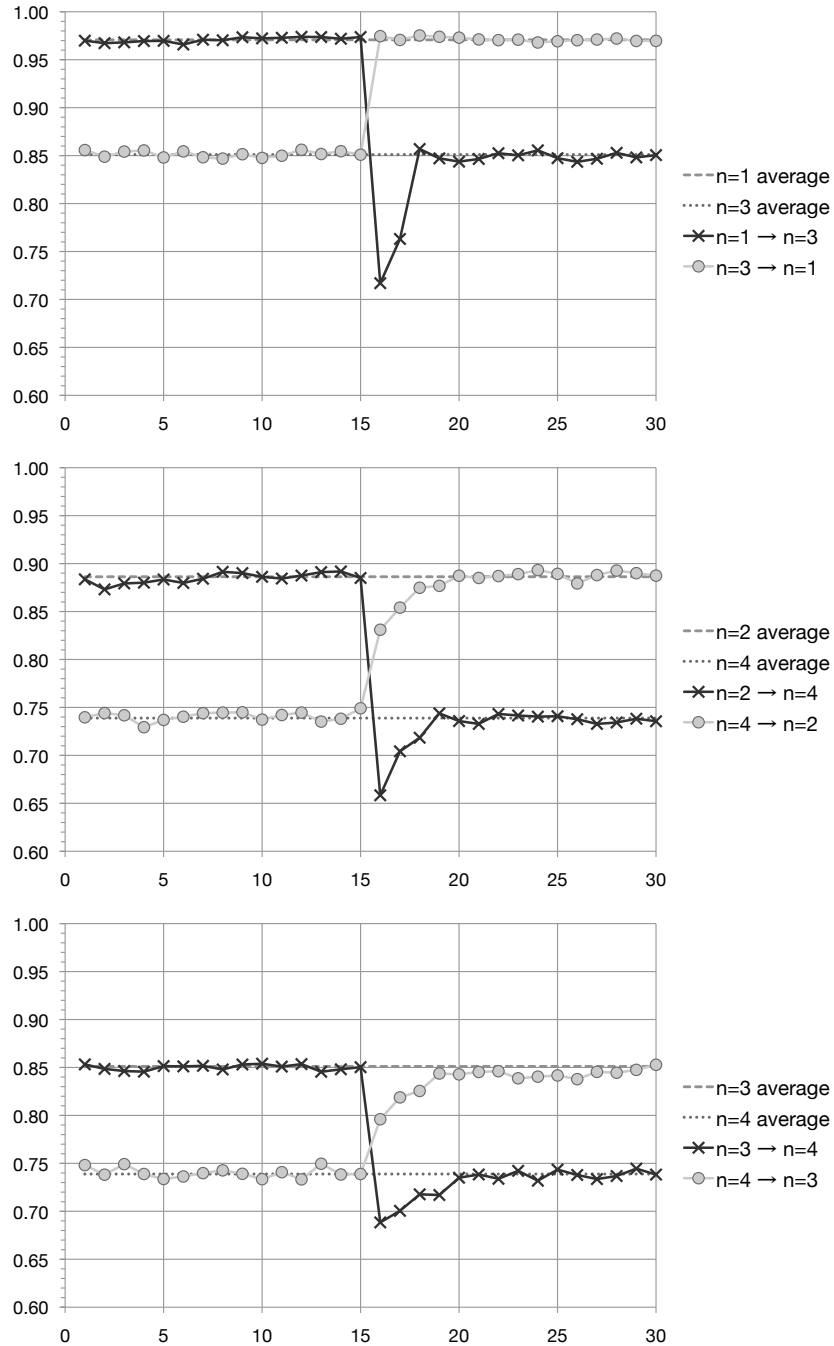


FIGURE 4.17. Accuracy when changing the value of n during a trial. The horizontal axis is the serial position within each sequence (following the preparatory period), and the vertical axis is the average accuracy of responses to that stimulus across all trials. Values of n were switched beginning with the 16th stimulus. [Top] Switching from $n_1 = 1$ to $n_2 = 3$ (dark crosses) and vice-versa (light circles). The average accuracy for the $n = 1$ and $n = 3$ conditions are shown as dotted and dashed lines, respectively. [Middle] Switching between $n = 2$ and $n = 4$. [Bottom] Switching between $n = 3$ and $n = 4$. (Note that the vertical axis does not begin at 0.0.)

averaged over 1000 trials. Prior to changing the value of n , the model performs as expected: at the average accuracy for n_1 . After the switch the model's accuracy is indistinguishable from trials in which the model was run at n_2 for the entire trial (call this the *baseline n_2 accuracy*). Thus there is no long-lasting performance penalty associated with having had to switch versions of the task. However, although the transition between values of n is quick, it is not perfect. Whenever n is decreased, there is a brief transitional period in which performance is below, but monotonically rises to, the baseline n_2 accuracy. This gradual increase in accuracy occurs because some patterns added to working memory are never unlearned during the transition period, resulting in increased interference. The only exception to this pattern is whenever transitioning to $n_2 = 1$, in which case the model's accuracy jumps to the baseline n_2 accuracy level as soon as the new value of n is input. We believe this is possible because the attractor for the 1-back stimulus is strong enough, having been decayed only once, to overcome any problems introduced by insufficient unlearning of other stimuli. In contrast, and unexpectedly, when n is increased there is a sharp drop in accuracy below the baseline n_2 accuracy. This occurs because when $n_2 > n_1$ some patterns which are needed have already been unlearned. This premature unlearning problem is only temporary, however, as the patterns which experience too much unlearning decay as more stimuli are added to working memory. After several stimuli the model is behaving as if it was never forced to switch between versions of the task.

In summary, the GALIS n -Back model makes several testable predictions about the average accuracy following a mid-trial change from n_1 to n_2 . First, after a brief transition period the accuracy is always the same as the baseline n_2 accuracy. Second, if $n_2 < n_1$, there will be a rapid monotonic rise in

accuracy to the n_2 baseline value. Third, if $n_2 > n_1$, there will be a sudden, sharp deterioration in accuracy below n_2 's baseline value, followed by a rapid monotonic rise to n_2 's baseline value. To our knowledge, data does not yet exist that can support or refute these predictions.

In order to investigate the sources of model errors, 100 runs of Model C were executed for $n \in \{1, 2, 3, 4\}$ according to the same procedures outlined at the beginning of this section. The $n=5$ case was not evaluated because it was already performing no better than chance, as explained earlier. For each time step, the action chosen by the control module was recorded. This was compared to the correct responses; for instance, for $n=3$, actions 1, 2, 5, 4, 3 (action numbers are listed in Table 4.4) should occur in that order for each stimulus. We then computed the Levenshtein distance⁶ between the actual and ideal responses (Navarro, 2001). A value of zero indicates a perfect match, meaning the control module never selected an incorrect action during that trial. The errors made by runs with non-zero distances can be attributed to failures of the control module. Errors made during a run with zero Levenshtein distance (no controller errors) were generally due to a failure of the working memory, such as a recalled pattern which is too noisy to be properly identified, or a failure to advance to the previously trained item. (It is possible that the

⁶ Levenshtein distance is a measure of string distance in which the two strings do not need to be the same length. It is a count of the number of symbols which must be added, removed or modified to produce one string from the other, and thus is a natural fit for this situation as we are interested in the number of actions which are missing, duplicated or erroneous. Formally, it is defined recursively as $\text{lev}_{a,b}(|a|, |b|)$ where

$$\text{lev}_{a,b}(i, j) = \begin{cases} \max(i, j) & \text{if } \min(i, j) = 0, \\ \min \left(\begin{array}{l} \{\text{lev}_{a,b}(i-1, j), \dots \\ \text{lev}_{a,b}(i, j-1), \dots \\ \text{lev}_{a,b}(i-1, j-1) - \delta_{ij}\} \end{array} \right) + 1 & \text{otherwise} \end{cases}$$

working memory could correctly recall a pattern and the compare module fails to correctly judge it, but testing the components individually revealed that this was hardly ever the case.) Both control and working memory errors are due to incorrect associations in the respective Hopfield networks, and can be linked to Hopfield nets' limited capacity and stochastic recall process (Ma, 1999; McEliece et al., 1987). It is possible that these errors could be reduced by employing different learning rules which have been shown to increase the capacity of Hopfield nets (Storkey, 1997; Storkey and Valabregue, 1999).

The proportion of runs having control errors can be seen in Fig. 4.18(a). As n increases so does the prevalence of control errors, since the length of the instruction sequence required is greater. The overall accuracy for all trials is compared to the accuracy only for those networks which made no control errors in Fig. 4.18(b). The proportion of responses which were incorrect as a result of malfunctions in the control module, working memory module, or both is shown in Fig. 4.18(c). Even though more errors can be made by the control module as n increases, the number of errors made by the working memory increases even faster. As a result, a larger proportion of errors can be attributed to mistakes in the working memory at higher n . Increasing error rates at higher values of n are to be expected since errors in both the control module and working memory can occur at any step during processing and higher values of n necessarily include more processing steps, allowing more errors to accumulate.

Runs in which the control module made errors can be subdivided into two groups: those making "pathological" and "non-pathological" errors. The pathological set were those that completely failed to output a particular action for the entire trial, for instance, a network which throughout was never able to enter the attractor state corresponding to action 4. These networks produced

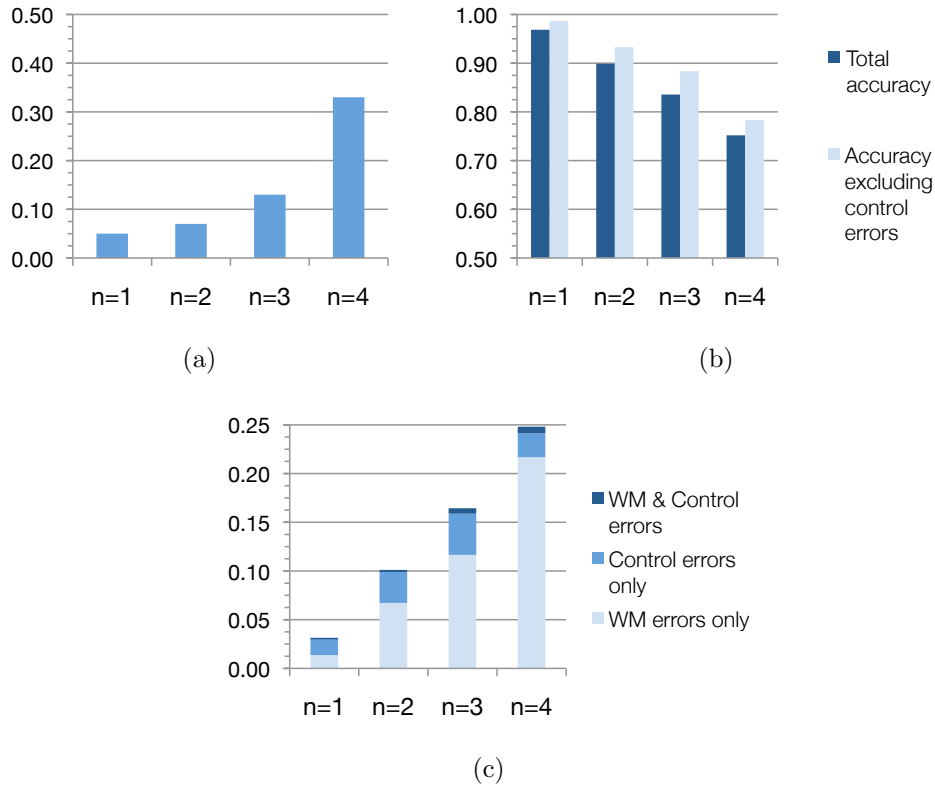


FIGURE 4.18. (a) The proportion of runs in which the control module made one or more errors for $n \in \{1, 2, 3, 4\}$. (b) The accuracy of model outputs for all runs (darker bars), and for those runs where no control errors were made (lighter bars). (Note that the vertical axis does not begin at 0.0.) (c) The proportion of model responses which were incorrect due to malfunctions in the working memory, controller, or both.

Levenshtein distances over 50. The non-pathological networks were those that made occasional errors, but were able to respond perfectly to a majority of stimuli. These networks tended to produce Levenshtein distances between one and ten.

We believe these pathological conditions are caused either because the attractor basin associated with the un-recalled action is either too small or too close in state-space to another basin. Either of these situations can occur simply as a result of having randomly chosen the bit patterns for the

internal representation of that action. This could be resolved by selecting semi-orthogonal patterns to represent each action, or ensuring there is a minimum Hamming distance between internal representations.

4.4 DISCUSSION

4.4.1 TESTING THE GALIS HYPOTHESIS

The work described here with the n -Back task demonstrates for the first time that the GALIS framework is capable of supporting executive functioning more typically associated with symbolic AI systems. This executive behavior allows GALIS-built models to exercise control over their own working memory. This control is a function not only of the structure of the network, as is usually the case, but also of the activity patterns learned by the instruction memory that make it possible to “program” a network to a novel extent. For example, after an initial training phase is complete, the GALIS n -Back model performs n -Back tasks for varying values of n without any direction from the user about how or when to modify its memory, activate outputs, etc. This independence is maintained even when the model switches between different versions of n -Back dynamically within trials. This was made possible by using attractor networks with asymmetric learning that control the sequential opening and closing of gated connections between model components. It was not necessary to rely on symbolic production rule-based systems, complex models of spiking neurons, locally encoded information, or biologically implausible learning rules. Parameters did not need to be re-tuned in order to match human performance on 1-, 2- and 3-back versions of the task, although adjusting the working

memory decay rate did lead to a better fit. There is also a theoretical argument for why lesser amounts of decay are desirable for larger values of n , namely, people may implicitly adjust decay in an attempt to change the size of their memory buffer (Altmann and Gray, 2002; Weems et al., 2009). The number of time steps the model takes for each stimulus is also highly correlated with the response times of human subjects. No modifications of any kind were necessary to capture this relationship in response times.

Importantly, testable predictions were made regarding 4- and 5-back as well as for intra-sequence changes to n . Although human participants find 4- and 5-back difficult, some studies test such lengths under certain conditions, and research is ongoing on training regimes which may allow humans to perform at such levels with practice (Harbison et al., 2011; Jaeggi et al., 2010). The method presented here for modeling n -Back could be falsified if human subjects failed to produce the observed patterns in accuracy changes when switching values of n within trials. I am not aware of any studies in which human subjects have been required to change n midstream, as opposed to between trials or blocks of trials, but if such studies were carried out one would expect to see the patterns evident in Figure 4.17.

One key point is that an important part of the behavior of the GALIS n -Back model, the value of n , is encoded in the contents of its adaptive instruction memory rather than the model architecture or in hand-coded connections. No changes to the model were required to perform five different versions of n -Back; changing the inputs to the model is sufficient to effect different behaviors in the model. Additionally, there are only a few major architectural differences between this model and a previously implemented model of the much simpler Store/Recognize task. These consist primarily of adjusting output — and to

a lesser degree, input — system to accord with the particular requirements of the task (e.g., two rather than three output nodes) and introducing one new component, the context module. This new component could be generally useful for a variety of tasks, not only n -Back, since it acts much like a very simple Program Counter in a CPU. In addition to the context module, the compare module present in both of these models is also a dedicated piece of architecture, hand-designed by the modeler. This does decrease the generality of the model, but I would note that this component is one which is also useful in a large variety of situations, and so while it is specifically tailored to one purpose, this purpose could generally be useful for a spectrum of future tasks.

4.4.2 GALIS AS A GENERAL PURPOSE FRAMEWORK

The behavior-as-software approach used in the GALIS framework is a novel approach among neural network models. In a sense, the ability to store temporal sequences of “instructions” in a control module and gate the activity throughout model regions based on these instructions gives GALIS models the ability to act in a “computer-like” fashion. Further, I predict that changing the information learned by the control module will enable GALIS models to be adapted to other tasks. In other words, I believe that building models of other similar tasks will require mostly changes to the neural software and only minimal changes to the neural hardware as I will demonstrate in the following chapters.

The use of both attractor networks and gating help to overcome one of the primary challenges of working memory: the need to balance stability with plasticity (Goldman-Rakic, 1987). Each pattern stored in both the working memory layer and the attractor networks of the controller gains stability from

being represented as the minimum of an attractor basin. However, the states of these networks can still be rapidly updated by introducing biasing inputs from external sources or adjusting nodes' thresholds. Similarly, gating can be used both to stabilize a network by, for instance, closing off its external inputs, or to rapidly destabilize a network by allowing inputs or triggering weight updates (O'Reilly et al., 2002). This is reminiscent of the D1 and D2 forms of attractor dynamics present in the prefrontal cortex (Durstewitz and Seamans, 2008). Attractors dominated by D1-type dynamics have deep basins, aiding robustness of working memory but increasing perseveration, while those dominated by D2 dynamics have shallow attractors, allowing fast switching and high flexibility, but making maintenance more difficult. Differing activation of the relevant dopaminergic systems can shift the attractor systems between modes, similar to the way that opening and closing gates governing biasing inputs can reform the attractor networks in GALIS.

Of course, the GALIS framework currently has some limitations, and will evolve in its details as it is used for additional tasks. These limitations include requiring learned instruction sequences to be determined by the modeler. Despite this issue, I think the current system of basing behavior on stored patterns in memory is a valuable stepping-stone towards more autonomous systems. If behavior can be stored in memory then it can be more easily modified than if it was built into the architecture. And if it can be modified, I believe it can be generated autonomously during learning. In other words, GALIS moves away from systems whose behavior is a function of their *construction* and towards ones whose behavior is based on *instruction*, with the eventual aim of not needing to provide those instructions explicitly. I am optimistic that the instruction patterns of the ISM can be modified online by the model because

the instruction memory operates by the same paradigm as the working memory layer, which I have shown can be modified by the model online. Future work needs to allow GALIS to modify instruction sequences during task performance, improving as it gains experience, and generate instruction sequences from the ground up, as is done in Chapter 6. This would enable GALIS models to proactively adjust their own behavior during trials rather than carrying out a predetermined sequence of reactions to the environment.

A second shortcoming of the current GALIS approach to storing instructions is the inability of temporally asymmetric attractor nets to store sequences in which the same item is repeated a given number of times without resorting to storing multiple tokens each representing the same type. There is a diverse assortment of attractor net methods for storing sequences (e.g., Farrell and Lewandowsky, 2002; Koene and Hasselmo, 2007) which are being explored to resolve this issue, in addition to looking into other neural approaches to serial memory (e.g., Botvinick and Plaut, 2006; Kremer, 2001; Monner and Reggia, 2012).

Finally, while GALIS is not intended as an accurate model of the brain, it is loosely inspired by the organization of cerebral cortex, especially frontal regions. For example, the control module's rule-like behavioral sequencing captures roles believed to be played by lateral prefrontal cortex (Bunge, 2004; Tanji et al., 2007), and the compare module's pattern matching activities can be related to the performance and detection of incongruent stimuli functions of the anterior cingulate cortex (Brown and Braver, 2005; MacDonald et al., 2000). An important direction for future research will be to further bring GALIS into alignment with known neuroscientific data.

Visuospatial Processing and Binding

5.1 INTRODUCTION

The cognitive control capability of the GALIS framework was evaluated in the previous chapter using two sequential memory tasks. Specifically, it was applied successfully to learn to perform simultaneously five versions of the classic n -Back task that is widely used in psychological testing. Not only did the resulting n -Back model function correctly, but its accuracy and response times correlated strongly with those of human subjects performing the same task (Sylvester et al., 2013). While this was encouraging, the n -Back task is relatively limited in terms of cognitive operations; for example, it does not involve any spatial information.

The goal of this chapter is to extend the GALIS framework to a much more challenging task that, for the first time, involves incorporating spatial relationships and addressing the binding problem concerning two different types of visual information (Feldman, 2013). Specifically, it presents a model for a card matching task in which an agent uncovers pairs of face-down cards, trying to select pairs that have matching patterns on their faces. This task makes

heavy use of visual information processing, working memory of the location of specific previously-uncovered cards, attention mechanisms, and binding of visual patterns to the spatial locations in which they occur. The trained card matching system addresses this latter challenge by taking inspiration from the ventral “what” and dorsal “where” visual pathways of the human brain (Baizer et al., 1991; Ungerleider and Haxby, 1994) and how they provide integrated information to prefrontal cortical regions. Binding the general location information provided by the dorsal pathway with the appropriate detailed object-specific information provided by the ventral pathway is a significant challenge (Reynolds and Desimone, 1999), and is a key focus of the task-specific model presented here. Computational experiments show that the GALIS card matching system not only performs the task successfully, but that it does so in ways that are again reminiscent of observations of human subjects performing this task.

Like the n -Back model, discussion of the methods used in this chapter has been divided into two sections. A comprehensive informal overview is provided in the next section, with further technical details being provided in Section 5.3. The final two sections cover the results of experiments with the model, comparing it to both a symbolic alternative and human subjects, and a discussion of the findings.

5.2 METHODS

Here I apply the GALIS approach for the first time to a more complex spatial memory task known by many names including “Pairs,” “Pelmanism,” and “Concentration,” but which I will refer to here as the “card matching task.”

It is played by first randomly placing several pairs of cards face-down on a tabletop. The player turns over two cards each round, one at a time, with the goal being to uncover matching pairs of cards so that they can be removed from the table. Play proceeds until all cards have been removed.

The requirements of the card matching task expand on the cognitive control tasks described in the preceding chapters by placing stimuli patterns in a spatial environment. That is, the model is not just attempting to remember a set of abstract stimuli in the æther, but must successfully bind together what was seen with where it was seen in the environment. The binding of multiple features is an ongoing challenge for neural models. This task also requires that a system make judgements about the contents of its own memory. In other words, it is not enough to just store a series of stimuli, but the model must be able to make strategic decisions based on inspection of its own memory of what cards have been seen previously and their locations.

In this implementation of card matching each card is 9 by 13 pixels. The backs are a uniform dark grey, and the fronts are bichromatic patterns such as horizontal stripes, crosses, or diagonals (see Figure 5.1). Depending on the experiment either four, six or eight pairs of matching cards (i.e., eight, twelve or sixteen cards in all) are arrayed on the table top, initially all face down. The images on the fronts of cards consist of monochromatic, low-resolution simplifications of national flags. For instance the **X**-shaped card in Figure 5.1 is derived from the flag of Scotland, while the striped card could alternately represent the flag of Italy, Ireland or France.

The GALIS model for card matching is composed of seven modules as illustrated in Figure 5.2. These are the Visual, Location, Object, Motor, Working Memory, Conflict, and Controller modules. Their functions are

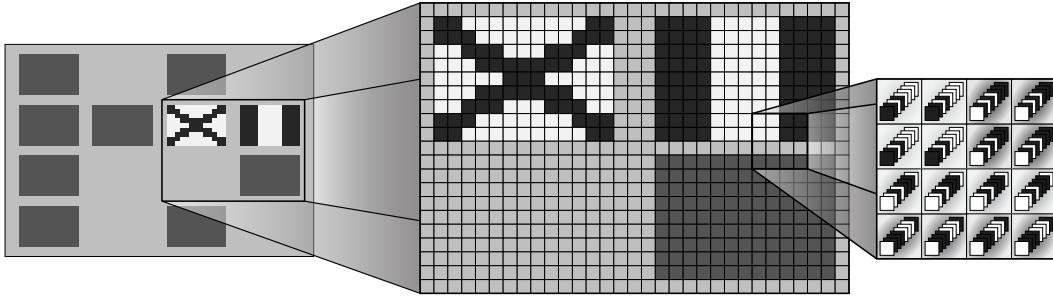


FIGURE 5.1. A depiction of the visual environment for the card matching task during play. On the left is a view of the full environment. Here there are 10 cards still present, two of which are face-up showing an \times and a vertically-striped pattern. The middle part of the illustration shows an expanded view of these two cards as well as one of the face-down cards. On the right a 4×4 section of the board has been blown up to show how each of the four “colors” used in the visual field is encoded as an 8 bit binary string. In this example, the white areas on the faces of cards (upper left of detailed area) are encoded as 11010000 and the black areas on the faces of cards are encoded as 0101111.

explained in the remainder of this section. Technical details can be found in Section 5.3.

5.2.1 VISUAL SYSTEM

The Visual System consists of the Primary Visual, Location, and Object modules. The Primary Visual module provides input to the model (Figure 5.3). Its visual field consists of a 55×67 grid of grayscale “pixels.” Each may take one of four values: light grey, representing the surface of the tabletop; dark grey, representing the back of a card; white and black, which together make the patterns on the front of the cards. Each of these values is represented by a random 8-bit pattern, so there is a total of 29480 nodes/neurons in the Primary Visual module.

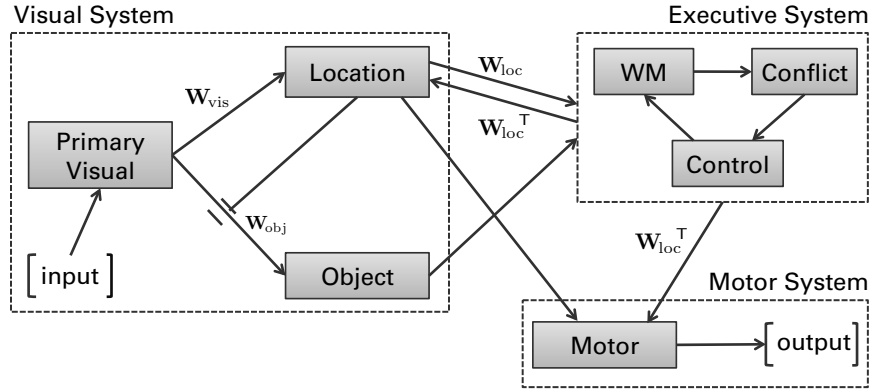


FIGURE 5.2. Overall architecture of the GALIS model for the card matching task. Its seven modules form three functional systems, the visual, executive and motor systems. Each system is indicated by an enclosing dashed-line box. See Figures 5.3 and 5.4 for more detailed views of the visual and executive systems, respectively.

The Visual module’s output is sent to two different regions for further processing. This is inspired by the parallel visual pathways present in the mammalian brain which connect the visual cortex to the prefrontal cortex via both the parietal/dorsal/where and temporal/ventral/what cortices. In this model, the Location and Object modules can be seen as simplified analogs to the parietal and temporal cortices, respectively (Baizer et al., 1991). The former is responsible for broad but low-resolution vision — identifying that there is an object at a particular location, but not particular features of that object — while the latter provides a detailed but narrow view — thus being able to discern details of an object but remaining ignorant of its location. The two visual pathways influence each other, with the Location module helping to guide the attention of the Object module, and the Object module providing detailed information about the visual field that the Location module lacks. This inter-pathway influence fits naturally with GALIS’ use of gating/higher-order network connections.

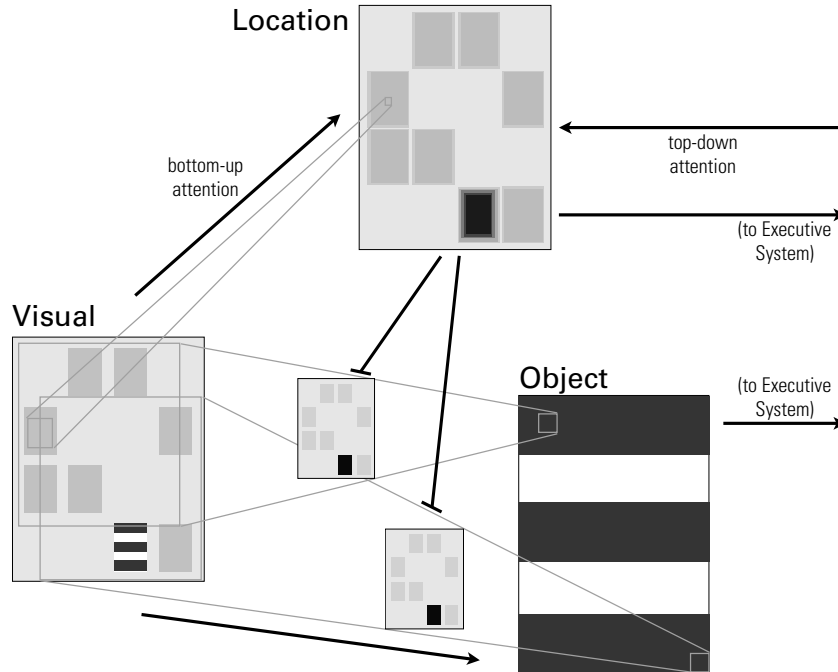


FIGURE 5.3. The model's visual system. Here there are 8 cards depicted in the environment. All are face-down except for one in the bottom row, which is striped. The region of the Location module corresponding to this card is the most active. A Location node along the left edge is also highlighted, along with its topographically corresponding receptive field in the Visual module. The state of the Object module reflects that it is attending to the sole face-up card. Two of its nodes and their receptive fields are shown. The smaller rectangles between the Visual and Object modules are depictions of the incoming weights to these two nodes, as set by the Location module. One can view these as a cross-section of the connections, with just one of the many incoming links to each node being open/active.

The Location module is the same size as the Visual module: 55×67 pixels. Rather than using 8 binary nodes to represent each pixel, each pixel is congruent with a single node with values in $[0, 1]$. This value roughly represents the salience the model gives to that location in the visual field. Nodes' activations are determined by the logistic sigmoid of the sum of both bottom-up input from the Visual Module and top-down input from the Executive System. For the

former of these inputs, each node is connected topographically, with a receptive field in the Visual module of a 5×5 square. The second input is from the Executive System. This input is gated so that when it is closed the attention of the model emerges from the interaction of the Visual and Location modules. When it is open, it serves to control visual attention from the top-down. This is of particular use when there are two cards face-up. Using only the bottom-up attentional mechanism both will be equally salient. The addition of the top-down element allows the Location module's focus to be directed to the card chosen by the Executive System.

The Location module also has two outputs. The first leads to the Executive system, providing it with a coarse view of the visual field so that it may determine object locations at a top level. The weights on these outputs, called \mathbf{W}_{loc} , form a random bipolar matrix of size 3685×1024 . This has the effect of assigning each node a random 1024 bit code. This pattern is then stored in the Working Memory to track where a card was seen.

The second output from the Location module is used, as mentioned previously, to control the receptive field of the Object module (see Figures 5.2 & 5.3). The Object module can access the visual field in finer detail, but at the cost of a limited scope. The Location module determines where the Object module should focus its limited field-of-view. I view this as an example of gating. The output of the Location module is able to open and close activity flowing from the Visual to the Object module. Thus, \mathbf{W}_{obj} acts like a mask, only allowing the portion of a Object node's receptive field which corresponds topographically to a card of interest to pass through. Activity in some portion of the Location module (which manifests as a rectangularly-shaped spike of activation) moves the focus of the Object module to attend to the area in the

visual field corresponding to the spike of activity. So, for example, if there is high activity in the upper left of the Location module then the connections between the upper left of the visual field and the Object module are opened and the rest are closed. This gating is essentially an example of higher-order nodes (Lipson and Siegelmann, 2000). The gating is implemented via “fast weights,” in which one network’s output (from the Location module) is used to adjust the weights of another network (the Object module).

The Object module is 9×13 pixels, each of which uses 8 binary nodes to encode “color” values, for a total of 936 nodes. Other sizes for the Object module were tested. Sizes smaller than the cards result in too much detail being missed and make it difficult to accurately judge whether two cards match each other or not. Tests with an Object module larger than the cards did not show any increase in accuracy, but did significantly increase computational overhead. Since the focus of this work is not principally about perception, I opted to make the Object module to be the same size as the stimuli.

The Object module is guided to focus on a particular region of the input plane by the Location module using a combination of the bottom-up information from the Primary Visual module and top-down information from the Executive System. Its output proceeds upstream to the Working Memory region with one-to-one connections, so that the model can form a memory of the visual appearance of cards it has seen.

5.2.2 EXECUTIVE SYSTEM

The Executive System consists of three modules (see Figure 5.2): Working Memory, a Controller, and a Conflict module. It is inspired by functionalities

generally associated with prefrontal cortical regions of the brain. The Executive System’s structure is shown in more detail in Figure 5.4.

5.2.2.1 WORKING MEMORY

The role of the Working Memory system is to store knowledge of which cards have been observed and where they were observed by integrating and learning the outputs of the Location and Object modules. This allow the model to choose pairs of cards intelligently based on its past experience (much as a person does) rather than blindly guessing at the locations of potential matches. The working memories used to remember external stimuli in previous GALIS models were unitary: they were capable of storing a sequence of binary patterns, but each pattern stood alone, without reference to any features such as its location in space (Reggia et al., 2009; Sylvester et al., 2010). Using the same approach that GALIS already uses to store instruction sequences, I now employ an auto associative network to effectively link representations of seen objects (i.e., overturned cards) and the locations at which they appeared.

Training of Working Memory is accomplished with standard one-shot Hebbian learning, which occurs whenever the Working Memory training gate is open. This establishes the learned pattern as an attractor in the Working Memory’s state space. This pattern can then be recovered when the network is in a state sufficiently close to it: either a noisy or corrupted version of the original pattern, or — more importantly for our purposes — when a part of the pattern is missing.

For example, if the X-shaped “Scotland” card depicted in Figure 5.1 is observed in the topmost row and leftmost column of cards in the environment then the working memory would learn a string corresponding to the tuple (X;

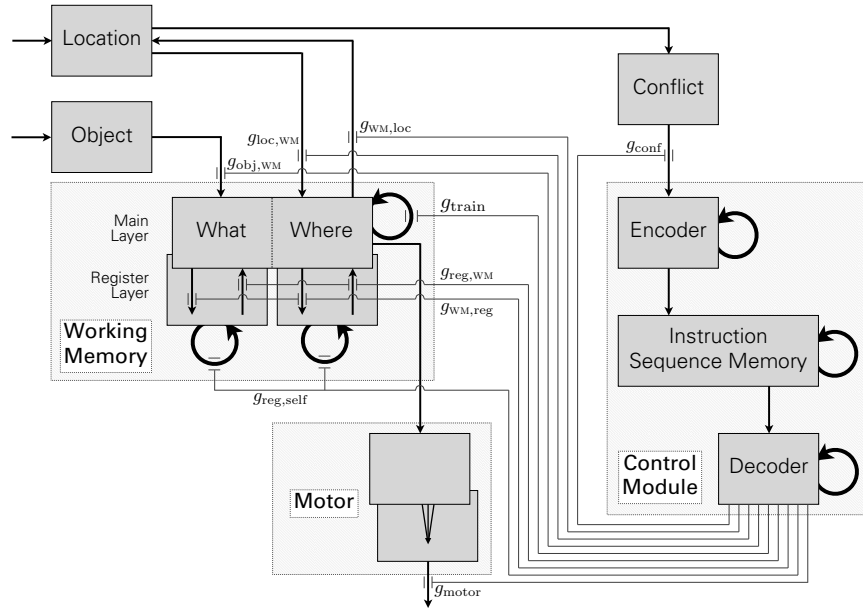


FIGURE 5.4. Architecture of the model's executive system. Thick arrows represent full internal connections. Thinner gray lines represent gating outputs from the Controller. Note that the size of layers is not to scale.

top-left corner). This string becomes an attractor state of the Working Memory network via one-shot Hebbian learning. This will allow the full pattern to be recovered from either portion. That is, by setting the “what” nodes to “X” and allowing the memory to update, the “top-left corner” portion can be recovered, and vice-versa. This is analogous to the way that the entirety of an image learned by an auto-associative memory can be recovered when presented with only a portion of the original image.

Every time the Working Memory is trained its prior weights undergo weight decay. This reduces the interference between patterns in memory, and allows for older memories to be supplanted by more recent ones. Adjusting the amount of decay can be used to affect the length of sequences that a memory like this stores: lower levels of decay allow longer series at the cost of more interference,

while higher levels of decay are better suited for shorter series (Winder et al., 2009).

The model's Working Memory also includes a "register" layer to allow the comparison of one pattern to another. These allow operations on multiple operands. The register layer is the same size as the main WM layer, and is linked to it by one-to-one, gated connections allowing patterns to be read into and out of this extra buffer.

5.2.2.2 CONTROLLER

The Controller is at the core of the Executive System. It is trained to direct the operation of the rest of the model by opening and closing the nine gates which govern the flow of activity, thereby allowing the entire system to function autonomously. The nucleus of the Controller is a discrete attractor network called the "Instruction Sequence Memory" (ISM). This is a memory unit whose purpose is to store the instructions for performing card matching rather than external stimuli as is the case with the main Working Memory component. As with the preceding chapters, this GALIS model's behavior is determined in large part by sequences of patterns that have been stored in its ISM, which shift the control of the network's behavior away from its architectural construction and towards its informational content.

Input to the ISM comes from a subcomponent called the Encoder, which is a hetero-associative network that is responsible for converting the Controller's input into a selection of which instruction sequence the ISM will process. The ISM's output is sent to the Decoder, which translates between the patterns stored in the ISM and the actual values which are sent to each gate. These hetero-associative sub-components serve dual purposes. The first is noise

reduction. This is possible because each specializes in either storing inputs or outputs, while the ISM is required to represent both inputs and outputs as well as the links between them. The second purpose is to decouple the size of inputs from outputs. For example, the Controller has nine outputs: one for each gate. However, more than nine nodes are needed to store a distributed representation of the six actions used for the card matching task in Hopfield-type attractor networks (Storkey, 1997). The Encoder and Decoder make it possible to translate between representations of differing sizes.

The structure and function of the Controller is unchanged from the GALIS instantiation of Section 4.3, where it was used to model performance of the n -Back task. I mention this not only to refer the reader to a more detailed description, but also to draw attention to the fact that the identical Controller architecture, when trained on a different set of instruction patterns, was used to perform this very different task—i.e., the controller provides a general purpose mechanism for cognitive control.

The ISM is trained prior to task execution on the necessary sequences of steps the model must take to perform the card matching task. Each of the eight rows in Table 5.1 are represented by a binary string, and each of these strings is stored as an attractor state of the ISM using one-shot Hebbian learning. These eight actions are linked together into three different sequences using temporally asymmetric weights. Each of these sequences correspond to the appropriate response to there being either zero, one or two cards face-up. Depending on the number of cards which are face-up—as judged by the Conflict Module—the Controller executes one of the three action sequences detailed in Table 5.1.

The ISM’s outputs are the nine gates distributed throughout the rest of the model (see Figure 5.4). They are:

1. the Working Memory Training gate (g_{train}), which controls when the weight matrix of the Working Memory layer undergoes training;
2. the Motor Output gate (g_{motor}), which allows the model to gesture to a location on the board to choose a card using the output of the Motor module;
3. the Location-to-Working Memory ($g_{\text{loc,wm}}$) and Object-to-Working Memory ($g_{\text{obj,wm}}$) gates, which allow the Working Memory state to be influenced by that of the current Location and Object Module states;
4. the Register-to-Working Memory ($g_{\text{reg,wm}}$), which governs the effect of the register on the “what” and “where” portions of the Working Memory layer;
5. the Working Memory-to-Register gate ($g_{\text{wm,reg}}$), which does the reverse;
6. the WM-to-Location gate ($g_{\text{wm,loc}}$), which allows the “where nodes” of the Working Memory to affect the Location Module, causing the Executive System to drive attention in a top-down way;
7. the Register-to-Self gate ($g_{\text{reg,self}}$), which allow the new register states to be dependent on their current state (opening this allows maintenance of a pattern, while closing it allows rapid updating), and;
8. the Encoder Update gate (g_{conf}) which governs the inputs to the control module, so that it can decide whether it begins a new sequence of instructions.

TABLE 5.1. Learned instruction sequences stored in the control module’s ISM.

Sequence	Action
1. Zero cards face-up	1. update WM to retrieve a stored pattern
	2. load WM contents into register
	3. load register contents into WM (excite object connections, inhibit location connections); enable top-down attention enable output from motor module
2. One card face-up	4. load object module contents into WM; load location module contents into WM
	5. load WM contents into register; train WM on current pattern
	3. load register contents into WM (excite object connections, inhibit location connections); enable top-down attention enable output from motor module
3. Two cards face-up	4. load object module contents into WM; load location module contents into WM
	6. train WM on current pattern

This set of gates is manipulated in order to act out the set of six different actions the Controller takes in execution of card matching. These six actions are combined in different ways to create the three sequences in Table 5.1.

When there are no card face-up in the environment, the Controller attempts to determine if it knows the location of a matching pair of cards. This is accomplished by retrieving a pattern stored in Working Memory and storing it in the Register. The Working Memory is then updated again, but its attractor landscape is perturbed by input from the Register. The connections between “what” nodes are set to be strongly excitatory, while those of “where” nodes are mildly inhibitory. This has the affect of causing the network to shift to a new attractor with the same “what” sub-pattern (i.e., representing the same card image), but a different “where” sub-pattern (i.e., known to be at a different

location). This “where” sub-pattern is then passed to the Motor Module for output and to the Location Module to provide the top-down portion of attentional control.

When one card is visible (face-up), the first step is to open the Location-to-Working Memory and Object-to-Working Memory gates so that the Working Memory’s state will represent the environment being witnessed. The WM Training gate is then opened, so that this observation is added to the Working Memory’s knowledge. Next, a check is run to see if there is a card in memory when matches the one currently being observed. This operates using the same method as in the zero-cards-up case, except here the card we are hoping to find a match for is the one which is currently visible rather than a random card chosen from memory contents as in the zero-card case. To accomplish this the visible card is stored in the Register, and the WM is updated with excitation on the “what” connections and inhibition on the “where” connections from the Register. If a second location for this card has been trained then it will become the new state of the WM network because the inputs from the Register are pushing the Working Memory state into the corresponding attractor basin. If a second location is not known there will be no basin in the region of attractor space the Working Memory is in, and so it will transition into some other attractor. While this attractor may be close in the state space of the WM network, from the point of view of locations in the external environment it appears to be a randomly guessed location. This has the desired affect of making the model guess a location to explore. In either case, the “where” sub-pattern which results from this update is passed to the Motor module for output and back to the Location module.

The model's actions when there are two cards face-up are very simple (as they are for a human player). If the two cards match then they are removed from the environment. If not, the only action required is to add the newly observed card to Working Memory. (Because the card which was already face-up was added to Working Memory when it was first revealed, i.e., while executing Sequence #2 in Table 5.1, there is only ever a single observation that needs to be remembered.) This is accomplished in the same way as with the beginning of the one-card-up case: open the gates allowing the Object and Location Modules to load their activity in the Working Memory and then update the Working Memory weights on this newly observed pattern.

5.2.2.3 CONFLICT MODULE

The Conflict Module's purpose is to gauge the level of disagreement in the Location Module about where to focus attention. This gives an indication of how many cards are face-up in the visual field. When no cards are face-up, the Location module will have minimal activity, resulting in very low conflict. When one card is face-up, it's location will be the single dominant source of activity, also resulting in low conflict. However, when two different card faces are visible, each location will vie for attention, causing internal disagreement about which to encode. The Conflict Module monitors for that disagreement, and reports it to the Controller. This behavior is inspired by the mismatch detection functions of the anterior cingulate cortex (Brown and Braver, 2005).

By informing the Control Module about how many cards are face-up, the Conflict Module allows the Controller to choose which of the three sequences described in the previous section it will execute. The connection from Conflict to Control Modules is gated; the signal to open the gate is only sent at the

termination of each of the three instruction sequences. This results in the gate being open the next time step, which allows the Controller to assess which sequence it should begin running. While this gate is closed, no input is received by the ISM, which allows it to continue running the current instruction sequence without interruption.

The inputs to the Conflict Module come from 512 randomly selected pairs of nodes in the Location Module, with each pair providing one bit of the eventual output. The desired output differs depending on the distance between the two nodes in a pair. Nodes which are topologically close in the Location Module should have similar states; indeed there is no inherent conflict in neighbors agreeing with each other. If nearby nodes have different states that is an indication of conflicting representations. In contrast, nodes which are far apart are not in conflict if they are both inactive, but are in conflict if they are both active as this represents the attempt to encode two disparate locations at once. The Conflict Module’s final output is effectively the proportion of pairs of nodes which are either nearby but in different states or far apart but both active.

5.2.3 MOTOR SYSTEM

The Motor System is intentionally very simple, consisting of only a single module (Figure 5.2), as detailed motor control is not being studied here. This Motor Module is roughly analogous to the premotor cortex. It consists of two layers (Figure 5.4), both of which are the same size as the Visual and Location Layers (55×67). The first layer has inputs from the Working Memory which are encoded using the same methods as the top-down mechanism linking the Working Memory to the Location module—i.e., $\mathbf{W}_{\text{loc}}^T$ (*cf.* Langner et al., 2013). The second layer is connected topographically to the first, with each

node being linked to 5×5 rectangle of nodes below it, each with a fixed and equal weight. This blurs the first layer’s representation of the output space, allowing a smoother and less volatile output.

When the controller has signaled that output should be allowed (i.e., by opening the output gate), the node with the maximum activity is interpreted as being a “gesture” to that particular location on the table top. If that node corresponds to an area in the visual field where a card is present, it is interpreted as the model “pointing” to that card, which is then “flipped” to reveal its identity.

5.2.4 EXPERIMENTAL METHODS

I compared GALIS experimentally to two other ways of performing the card matching task. The key measure of performance used is the number of rounds required to remove all cards. First, in order to assess GALIS’ similarities to and differences from people, data was collected from human subjects as they performed a web-based version of the card matching task that I developed.¹ The 34 participants played a total of nine times, three each with either 8, 12 or 16 cards on the board. This gave us 102 recorded trials for each of the three conditions. The ordering of trials was randomized for each subject to minimize biases due to ordering effects. The images used on the human subjects cards were randomly selected each trial from 10 pairs of national flags. To remove any potential influence of disparate hues and to better match the monochrome inputs of the neural model, all of the flags were composed of red, white and blue only (e.g., those of the United Kingdom, the United States, etc.).

¹ This can be accessed at <http://www.jsylvest.com/cards/>.

The second point of comparison for GALIS' neurocomputational approach is a top-down symbolic algorithm. To achieve this, I implemented a symbolic AI system to play a simplified version of the card matching task. In this AI system I have removed all the aspects of vision and spatial processing, and instead represent each card as a pair of integers: one for the location of the card, and one for the pattern on the card. The symbolic model pursues the following strategy: At the start of each turn, the model checks to see if it knows where a matching pair of cards are. If it does, that pair is removed from the board. If not, it randomly selects a card from a location which is not in its memory. If the location of the matching card is in memory, the pair is removed, otherwise a second random location is chosen (and if by chance the two randomly selected cards match, they are removed). Any time the model sees an overturned card, the card is added to memory.

In order to make the performance of the symbolic system more comparable to that of humans, I introduced a modifiable decay factor to its memory. On each turn of play, items in memory may be deleted with a probability equal to a decay rate δ_S . When $\delta_S=0.0$ there is no decay, and the symbolic model plays perfectly. (That is, on average does as well as is theoretically possible given random card placement and selection). When $\delta_S=1.0$ the symbolic model has no memory at all, and plays by random guesses. Intermediate values of δ_S allow us to produce intermediate behaviors, while extreme values allow us to compare GALIS to theoretically optimal performance ($\delta_S=0$) or random performance ($\delta_S=1$).

The GALIS results presented below are averages from 200 simulation runs of the model. In each case the model's ISM was pre-trained on the necessary instructions, which were identical for all three task variants. With 8 and

12-card variants, the locations of the cards was randomly chosen from among the positions used for the 16-card case.

5.3 FURTHER DETAILS OF THE CARD MATCHING MODEL

Because one of the aims of GALIS is to enable the construction of generalizable models that do not need major architectural changes to complete different tasks, many of the details of the model constructed for Card Matching are unchanged from those used for n -Back. (See Sections 4.3.1 and 4.3.2, as well as Sylvester et al. (2013).) Elements which have been changed or added, such as those used for visuospatial processing, are covered below.

5.3.1 LOCATION LAYER

The weights on the bottom-up connections (\mathbf{W}_{vis}) from the Visual to Location modules are trained using one-shot Hebbian learning. It uses extensive weight-sharing, so that every node has a different receptive field but identical incoming weights. Each node has the same role: recognize if its receptive field is “interesting” — i.e., it is looking at a portion of the front of a card, as opposed to the back or the table surface. Because each node has the same purpose, each node can have the same weights. So, for example, the weight on a connection to a node in the Location module from the top-left node in its receptive field is the same regardless of which node is being considered or where its receptive field falls in the Visual layer. This makes training much more efficient. The training patterns are a selection of possible 5×5 patterns which appear on

cards (12.5% of the total possible patterns are used for training). Through this process nodes learn to turn on when they detect patterns from the faces of cards, and remain off when their input field is the table top. This produces a rectangular surge of activity in the Location module which corresponds topographically to that of face-up cards in the Visual module. (In Figure 5.3, this is the horizontally-striped card in the center-right of the bottom row.)

The weights on these top-down connections are merely the transpose of the weights on the counter-flowing, bottom-up connections discussed immediately below. The final state of the node is just the sum of both the bottom-up and top-down influence, weighted by the appropriate gating factor.

$$l_i = \sigma (\mathbf{W}_{\text{vis}}^T R(i) + g_{\text{WM,loc}} \mathbf{W}_{\text{loc}}(:,i) \vec{a}) \quad (5.1)$$

Here l_i is a node in the Location Layer, $R(i)$ is the receptive field of node i , $g_{\text{WM,loc}}$ is the gate governing top-down attention, $\mathbf{W}_{\text{loc}}(:,i)$ is the set of weights out from node i to the Working Memory, and \vec{a} is the state of the nodes in Working Memory which store location data.

The output from the Location module to the Executive system is thus the average of each node's code string, weighted by the nodes' activity. As a result, overlapping spikes of activity produce similar outputs, despite the randomness of each individual node's representation. This system also has the desirable by-product of reducing the dimensionality of the spatial encoding from that needed by the Location module (3685) to that used by the Working Memory (1024). The overall effect is similar to that of Random Matrix Transformations (Achlioptas, 2003; Johnson and Lindenstrauss, 1984; Rahimi and Recht, 2007, 2008). Finally, this approach has the added advantage of being easily invertible: \mathbf{W}_{loc} is used to translate between the encoding used the

Location Module to that used by the Working Memory, providing bottom-up visual attention; $\mathbf{W}_{\text{loc}}^{\top}$ is used for the reverse translation, allowing for top-down attention control.

5.3.2 OBJECT MODULE

Every node o_i in the Object module has a receptive field of 47×55 nodes in the Visual module. At any one time, each Object module node should only be accepting input from one of those Visual module nodes. Furthermore, each node should be accepting input from the node in the same location in its field—i.e., if one node is attending to the middle of the top row of its receptive field, so should the others. Each spike of activity in the Location module therefore translates into a single active point in a 47×55 grid. These pairs of active regions in the Location module with their correlating points of focus in the visual field are used as training patterns for the hetero-associative Hebbian learning which is used to actually form the weights \mathbf{W}_{obj} controlling the Object module’s focus.

The activation of Object module nodes can be formalized as

$$o_i = \sigma \left(\sum_{j \in N(i)} \left(x_j \sum_{k \in \text{loc}} \mathbf{W}_{\text{obj}jk} l_k \right) \right) \quad (5.2)$$

where σ is the logistic sigmoid function, N_i is the receptive neighborhood of node i , x_j is the state of a node in the Visual module, loc is the set of nodes in the Location module, l_k is the state of one of the nodes in the Location module.

5.3.3 WORKING MEMORY

Nodes in the registers update according to the simple rule:

$$r_i = \text{sgn}(g_{\text{reg,self}} r_i + g_{\text{WM,reg}} s_i) \quad (5.3)$$

where s_i is the state of the topographically corresponding node in the primary Working Memory layer. It can be seen that the new state of a register is either the persistence of its current state or a switch to the state of the primary WM layer, depending on whether the register-to-self gate ($g_{\text{reg,self}}$) or the WM-to-register gate ($g_{\text{WM,reg}}$) is open. That is, depending on the the gate signals the register will either maintain the current state, or load a new one from WM. This crystalizes the dichotomy between stable maintenance and rapid updating (Goldman-Rakic, 1987).

The updated states of nodes in the primary WM layer are the result of a sum, weighted by the appropriate gate values, of the current state, the state of the register, and the output of the visual system. For nodes which encode “what” information, this latter value is the simply the state of the Object Layer. For nodes encoding “where” information, it is the state of the Location Layer, as weighted by the \mathbf{W}_{loc} weight matrix. This can be formalized as

$$s_i = \text{sgn} \left(\sum_{j \in \text{WM}} \mathbf{W}_{\text{WM}ij} s_j + g_{\text{reg,WM}} r_i + g_{\text{obj,WM}} o_i \right) \quad (5.4)$$

for “what” nodes and

$$s_i = \text{sgn} \left(\sum_{j \in \text{WM}} \mathbf{W}_{\text{WM}ij} s_j + g_{\text{reg,WM}} r_i + g_{\text{loc,WM}} \sum_{k \in \text{loc}} \mathbf{W}_{\text{loc}ik} l_k \right) \quad (5.5)$$

for “where” nodes. Here o_i is the i th node of the Object module, l_k is the k th node of the Location module, $g_{\text{reg,WM}}$ is the gate on the register-to-WM

connection, $g_{\text{obj,WM}}$ is the gate on the Object-to-WM connection, and $g_{\text{loc,WM}}$ is the gate on the Location-to-WM connection.

5.3.4 CONFLICT MODULE

The amount of conflict present in the Location module’s activity is estimated based on a sampling of the conflict between 512 pairs of nodes. It is of course possible to define the overall conflict to be a function of the entire module’s state, but this global calculation is both computationally expensive and unnecessary. Only a small fraction of the pairwise conflicts are needed to get an accurate assessment of the number of locations that are presently being represented.

The desired output of a node in the Conflict Module differs depending on the topological distance between its two input nodes in the Location module. Nodes which are close should be expected to have similar states; indeed there is no inherent conflict in neighbors which are both on or both off. Nodes which are far apart are not in conflict if they are both inactive, but are in conflict if they are both active as this represents the attempt to encode two locations at once. The goal then, is to output $x_1 \oplus x_2$ if nodes i and j are within some topological distance d of each other, and to output $x_1 \wedge x_2$ if they are not.

To accomplish this, each pair of nodes is connected by a network like that shown Figure 5.5, with the weights w_{AC} and w_{BC} set according to the distance between x_1 and x_2 . (We use a Chebyshev distance equal to 7.5 to differentiate between “near” and “far.”) More formally, if $\|i, j\|_\infty > 7.5$ then $w_{\text{AC}} = 1$ and $w_{\text{BC}} = -1$, otherwise $w_{\text{AC}} = 0$ and $w_{\text{BC}} = 1$.

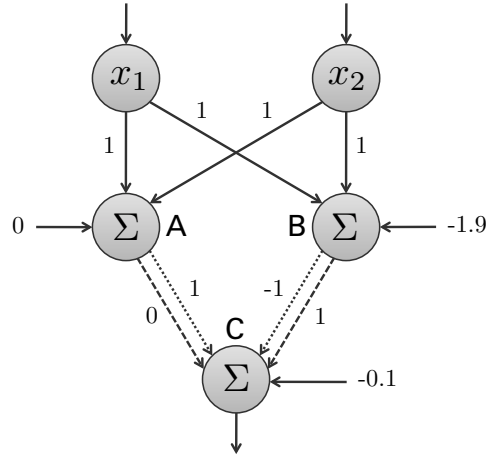


FIGURE 5.5. Wiring of a pair of nodes in the conflict module. There is only one link from A to C and one from B to C, with the weights on these links depending on the distance between x_1 and x_2 . The weights labeled on the dashed lines are used for nodes near each other, while the weights on the dotted lines are used for nodes which are far.

5.4 RESULTS

The decay rate of the symbolic AI model was calibrated prior to evaluating this GALIS model. The effect of decay rate on this model's performance can be seen in Figure 5.6. In all three task versions, both the average and the standard deviation of the number of rounds needed to complete the task increased superlinearly with the decay rate. Results from 200 runs of the symbolic model showed the closest fit to human subjects for the $n=8$, 12, and 16-card conditions when the symbolic decay rate $\delta_S = 0.475$, 0.40, and 0.30, respectively. This is consistent with analogous findings in past computational studies where decreased decay was correlated with increased working memory span (Altmann and Gray, 2002; Reggia et al., 2009; Winder et al., 2009).

The GALIS system successfully solved every card matching task on which it was tested. The number of rounds it needed to complete the task, averaged over

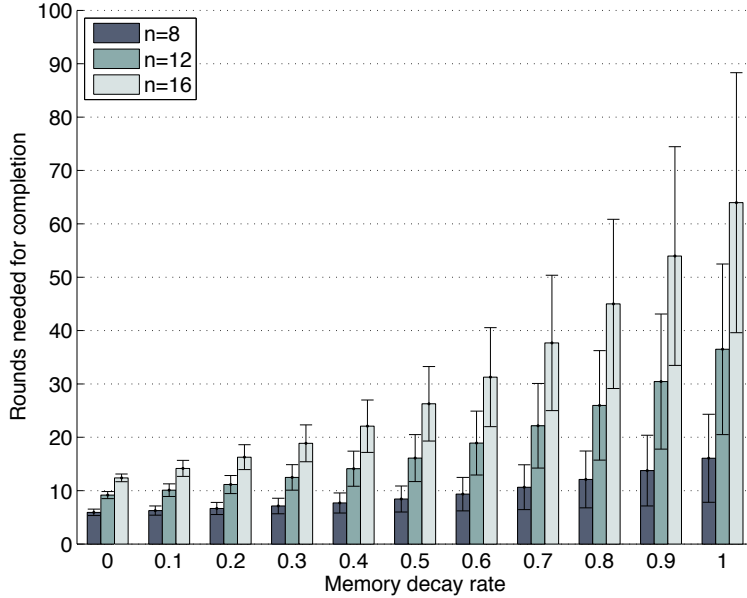


FIGURE 5.6. Performance of the symbolic model on all three card matching task versions with varying decay rates. Note that as decay increases, both the expected performance as well as deviation in performance increases. Based on 200 runs of the symbolic model for each value of the number of cards n . Error bars represent standard deviations.

200 runs, was 8.7, 13.0 and 21.2 for 8-, 12- and 16-card versions respectively. This compares to mean human scores of 7.9, 13.5 and 18.9. This was achieved with a decay rate $\delta_N=0.125$, so it was not necessary to adjust the parameters, structure or instructions of the model in any way to perform in all the three conditions. If the symbolic model was similarly limited to a single choice of decay parameter δ_S then there was no significant performance difference between it and the neural methods used by GALIS. This was possible despite operating in a much more complex environment than previous GALIS models.

Results with GALIS (200 runs for each value of n) are shown in Figure 5.7. These are compared with both the results from human subjects and from three instances of the symbolic model—one with no memory decay ($\delta_S=0$), one

without any memory ($\delta_S=1$), and one with $\delta_S=0.375$, which was the decay value which provided the best fit to the human results across all three task conditions. The mean number of rounds it took GALIS to complete the task increased as n increased, consistent with human performance and with expectations. We find no statistically significant difference at the $p=0.05$ level between the GALIS model’s performance and that of human subjects or the best-matching symbolic model on both the 8- and 12-card conditions. The GALIS model performed somewhat worse than humans on the more challenging 16-card version, as did the best-matching symbolic model. In this task condition the neural model slightly out-performed the symbolic model, but not at a significant level.

As expected based on the symbolic model and previous studies of attractor network-based working memory, a decrease in working memory decay rate δ_N was helpful as the problem size grew larger. The GALIS models were able to match human performance on $n=8$ and $n=12$ with a decay rate of 0.125 — i.e., adjusting this parameter was not necessary to fit data from both task versions — but optimal performance was observed when $\delta_N=0.15$ and 0.10, respectively. That is, a marginally lower decay rate increased memory capacity to allow for additional cards to be recalled. The associated and unavoidable trade-off is that reduced decay leads to increased interference between items in memory. The best performance on the $n=16$ condition occurred with $\delta_N=0.025$. This low level of decay was still unable to increase capacity sufficiently to match the human responses. Any lower values lead to dramatically more interference and worse performance, while higher values produce too much decay and worsen performance.

In order to investigate the causes behind the GALIS model’s less accurate match to human performance levels under the most challenging task condition

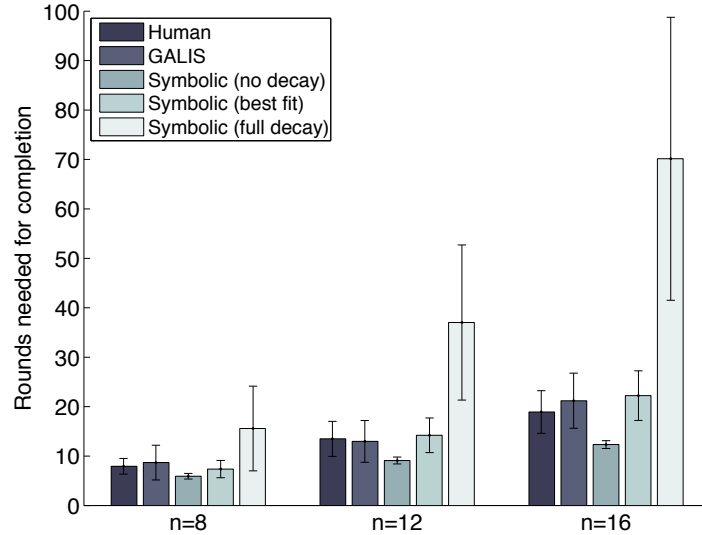
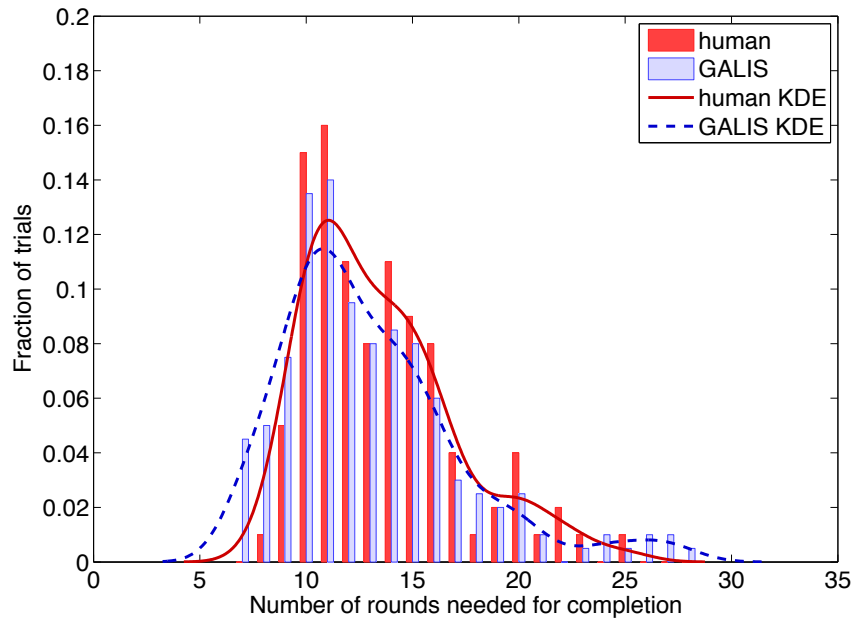


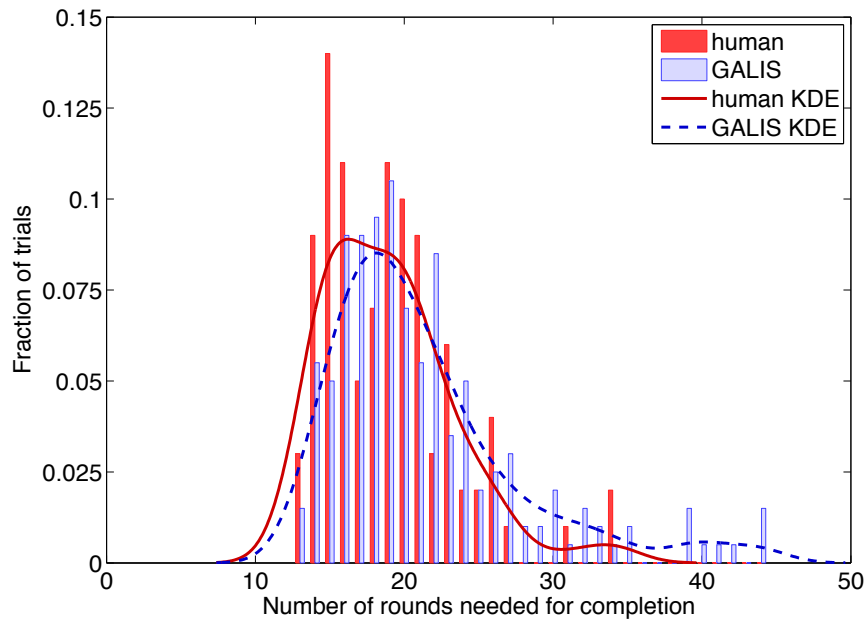
FIGURE 5.7. Mean and standard deviation for human subjects, as well as symbolic and GALIS models on the card matching task for the $n=8$, $n=12$ and $n=16$ conditions. For all three n conditions the decay rate of the GALIS model was 0.125. Results from the symbolic model are shown when it experiences no decay to its memory, a decay rate of 0.375, and complete memory decay. By adjusting the decay rates of both models it was possible to produce better fits to the human data, but the values used here provided the best fit across all three n conditions without varying the decay rate. The difference between the human results and those of the GALIS model and the best-fitting symbolic model are significant only in the $n=16$ condition.

we constructed histograms of the performance for both humans and the neural model. These, along with a kernel density estimate (KDE) for smoothing, are shown in Figure 5.8. As can be seen in the right subplot for $n=16$, the difference between human and GALIS performance is largely due to a thicker right-hand tail on the distribution of GALIS results. Without these few outlying runs, which required over 40 rounds to complete, there was no statistically significant difference from the human results.

Figure 5.9 shows a similar plot with the bars omitted for the $n=16$ variant, with human performance, GALIS results, and results from the symbolic model



(a) $n=12$



(b) $n=16$

FIGURE 5.8. Histogram of human (red) and GALIS (blue) performance on 12-card and 16-card task versions. Also given is a curve showing a smoothed estimation of each histogram using gaussian kernel density estimates (ODE). Human results are the solid red lines, and GALIS results are dashed blue lines.

with two different decay rates. When $\delta=0.35$ there was no significant difference between the performance of the symbolic and GALIS models, and both were worse than the human level. The symbolic model was able to decrease the numbers of rounds needed by lowering its decay rate to 0.3. This slight parameter shift was all that was needed to cause the symbolic model to go from matching the GALIS model to the human performance level for $n=16$ (but not for other values of n). This indicates a partial cause behind the GALIS model’s inability to match human results on this task version: GALIS’ decay rate was already set very low, making further decreases futile.

It helps to understand the causes of these worst performers — as well as why they have a significant effect only when $n=16$ — to consider what happens if locations are guessed completely at random. The chance of randomly picking a matching pair of cards in any given turn in which there are c cards on the board is $\frac{1}{c-1}$. The number of turns needed to randomly uncover the first matching pair is thus given by a geometric distribution with $p = \frac{1}{c-1}$, which has an expected value of $1/p = c - 1$. Once the first pair is found, the number of cards decreases by two, and the process is repeated. An entire game of matching n cards without any memory can be modeled as the sum of a sequence of geometrically distributed random variables.

$$X = \sum_{i=1}^{n/2} X_i, \quad X_i \sim \text{Geo} \left(p = \frac{1}{2i-1} \right) \quad (5.6)$$

As a result of this process, adding an extra pair of cards causes the expected number of rounds needed to complete the task to grow quadratically. Importantly, it also causes the standard deviation of the rounds needed to grow quadratically. Not only does the average number of turns increase with number

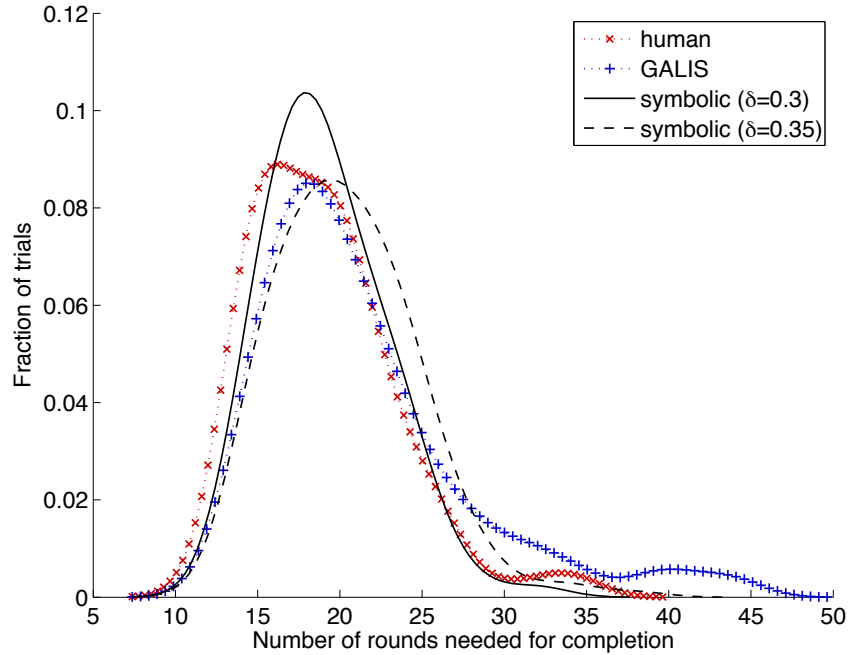


FIGURE 5.9. Kernel density estimates on the 16-card task for human subjects and the GALIS model compared to the symbolic system with two different decay rates. When $\delta_S = 0.3$ the symbolic system and human performance match, and both outperform the GALIS model. Increasing the symbolic system’s decay rate to $\delta_S = 0.35$ shifts its performance curve to the right, causing it to be statistically similar to that of the GALIS model. Note also that the performance distribution of the symbolic model displays the same positive skew as do humans and GALIS models, and that the skewness increases with higher decay values.

of cards, but the chance of a poor-performing outlier increases greatly as well. This is the pattern we observed when moving from $n = 12$ to $n = 16$.

The assumption that a player’s choices are being made randomly is, of course, incorrect. But note that the player will behave more like a random guesser in early turns, since little is known about the cards. It is exactly those early turns when the most cards are still present—i.e., i is close to $n/2$ —that will dominate the series of random variables X_i above. Furthermore, the more

rounds that are played the more patterns that will be added to working memory, and the more the limited capacity of working memory becomes a constraint. Poor initial performance in a run can create a feedback loop: as more turns elapse, interference between patterns in working memory increases, causing difficulty in recall, causing more turns to elapse, and so on. Small perturbations in the agent's working memory capacity may therefore result in a very different distribution of outcomes due to this positive feedback. The GALIS networks had the decay in their working memory set as low as possible for the $n=16$ task. This produces a concomitant increase in interference, causing them to behave more like a randomly-guessing agent.

Figure 5.10 shows an observation of this pattern. The blue area in the middle of the plot shows a non-parametric estimate of the average performance of 100 simulations with $n=16$ (Hsiang, 2013). Two particular runs of the GALIS model are shown in red. The dotted line shows one run with a final score of 17, while the simulation represented by the solid line took twice as many turns to finish. The difference is entirely due to the inability of the latter to find the first matching set of cards among the 8 pairs on the board. After this hurdle is cleared the remaining pairs are identified even more rapidly than they are in the high-performing example. This early plateau pattern was characteristic of the few poor-performing simulations that made the GALIS model's results not precisely match those of human subjects when $n=16$. Examination of these outlier runs showed that the controller was working precisely as it was trained to do, but that by chance the same location cards were frequently being re-picked early on. In other words, the algorithm in Table 5.1 does not adequately anticipate this possibility, allowing it to occur in a few percent of

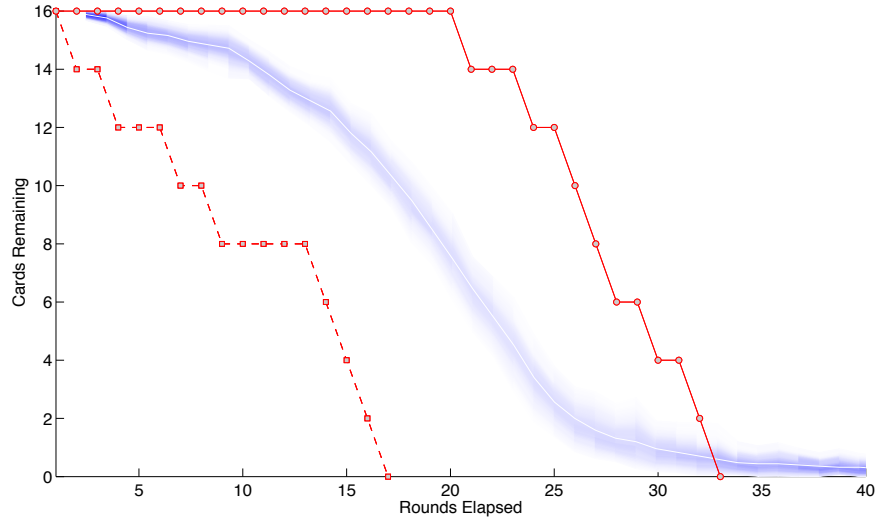


FIGURE 5.10. A visually-weighted regression of the model’s average performance over time in 100 runs of the 16-card task variant (blue), along with the performance curves of one high and one low-performance runs (dotted and solid red, respectively). The shading represents the width of the confidence interval surrounding the performance, as determined by a nonparametric bootstrap estimate (Hsiang, 2013).

the simulations and thus biasing the model’s performance overall to take a bit longer than humans do in this case.

Because the model has no trouble performing as expected once clearing the early plateau, we do not believe the Control Module is the cause of this behavior. (If it was to blame, we would expect the incorrect behavior to persist throughout the run.) In order to verify this we compared the action chosen by the control module in each time step to the action it should have taken in that situation. The model should execute actions #1, #2 and #3 when there are no cards face-up, then #4, #5, and #3 when one card is face-up, followed by #4 and #6 when two cards are visible, at which point the cycle should repeat. We were able to construct a list of the actions actually chosen by the Controller by recording which action’s representation was closest to the

state of the ISM at each time step. To do so, we compared the state of the ISM to the representations of the rows of Table 4.1. We then calculated the Levenshtein distance (Navarro, 2001) between the list of the correct actions and the actions chosen by the model.² A Levenshtein distance of zero indicates a perfect match between two strings, meaning the Controller never selected an incorrect action during that trial. Non-zero values were very rare in the sample. In the 100 simulations shown in Figure 5.10, only six occurred, and the maximum Levenshtein distance observed was four.

This leaves two potential, and related, sources of error. The first is typical Working Memory errors: the model incorrectly remembers stimuli by falling into spurious attractors or otherwise returning an erroneous pattern as a result of attractor networks' stochastic updating process (Ma, 1999). The second is a problem with the algorithm “programmed” into the model itself—i.e., programmer error. Qualitative analysis of the runs with low-performance indicate that there is a preponderance of the latter. There is a specific undesired behavior that recurred: the model would select from the same small set of locations for many rounds consecutively rather than exploring the full set of cards. Each time a card is turned over the Working Memory is trained to remember what was witnessed and where. But repeated exposures to the same cards causes the associated basins of attraction to grow larger and larger, crowding out the areas of the state space accorded to other locations. This results in it being more likely that those repeated cards will be chosen again, as they now have larger basins of attraction than before. This also creates a feedback loop from which the model has difficulty breaking out.

² See Chapter 4, note 6 *supra*.

5.5 DISCUSSION

In this chapter, I provide a substantial test of the basic GALIS hypothesis by applying it to a challenging card matching task. Performing this task requires, among other things, the ability to deal with representing external entities and their locations in space, the ability to support a robust working memory of previously seen cards, the ability to bind together distinct pieces of information about the environment, and the ability to exert top-down attention control and action selection. Such abilities are readily achieved with traditional top-down AI symbolic systems, but have proven to be extremely challenging for neural architectures (Martinet et al., 2011; Trullier et al., 1997), and go far beyond what has been attempted with GALIS previously.

The results presented here provide significant support for the GALIS hypothesis. Specifically, not only could the neurocognitive architecture learn to perform the card matching task, but the number of steps (card selections) it made during problem solving qualitatively increased with problem difficulty in a fashion similar to that seen when we had a group of human subjects solve this problem. At times, the model’s performance and the performance we observed with the human subjects we studied matched quantitatively. Where this match deviated significantly (with the larger number of cards), analysis indicated the most likely reason for this deviation was the author-generated instructions the system was trained to perform not some failure of the underlying principles. We also examined the influence of working memory decay on the system and found that while adjusting the rate was not required in order to match the behavior of human subjects, doing so did allow closer fits. Decreasing working memory decay allowed for handling larger number of stimuli, which corrobo-

rates previous studies of working memory decay as well as corresponding to the effect of varying decay rates on the behavior of the symbolic model we built. At the most challenging task condition, a zero lower bound was reached with the decay rate, making further increases in the model's memory capacity infeasible.

This neurocognitive system is composed entirely of components based on neural network methods that use distributed sub-symbolic representations. The finding that this system can perform cognitive control operations of the sort performed by traditional AI symbolic problem-solving methods is both encouraging and highly significant. Such cognitive control abilities are widely recognized to be challenging for neural computational methods. In a sense, this approach provides a synthesis of continuous neurocomputational representations and symbolic AI representations. Even though the GALIS approach uses only neural information processing, the fact that it allows one to “program” a neural network with a sequence of high-level instructions creates a similarity to the traditional von Neumann architecture computer. Further, even though the attractor networks operate in a high-dimensional, continuous state space, each attractor within that space exists as a discrete entity (Simen and Polk, 2009), and the use of gating allows for hard-cutoff binary distinctions to be made (open vs. closed communication channel, active vs. quiescent region, update working memory vs. maintain its current contents, learn vs. don't learn, etc.). As a result, GALIS networks offer a balance between the continuous nature of neural networks and the discrete nature of symbolic systems. Gating also has the further benefit of providing a way to balance the dual needs of maintaining stability of a network's state and for being able to rapidly switch

states (O’Reilly et al., 2002), a key requirement that has long been recognized as important in biological cognitive control systems (Goldman-Rakic, 1987).

The work here can be compared to several past related studies of neural systems for working memory and cognitive control. As described in Chapter 3, neural networks have been widely used to model cognitive control (e.g., Botvinick and Plaut, 2006; Kaplan et al., 2006; Pascanu and Jaeger, 2011; Ponzi, 2008; Verduzco-Flores et al., 2012). Many of these, such as C-SOB (Lewandowsky and Farrell, 2008), concentrate almost exclusively on the working memory aspect of cognitive control and rely on the modeler to make decisions about when to update weights or how to produce output. They also interact with their environments in a very limited way in the sense that they are exposed to stimuli in a set order, and produce one response in reaction to each — often yes/no; rarely are there more than four discrete answers to choose from — at which point the next input is provided regardless of the accuracy of the response. The GALIS model presented here must reach its own decisions about which of up to 16 cards to observe when, and the environment in which it must perform the remainder of the task is heavily influenced by its prior performance.

In many past models, the issue of appropriately binding a stimulus to the conditions of its observation is often side-stepped. Part of the stimuli provided to models is often constructed explicitly to contain the relevant information on time or space (e.g., Chatham et al., 2011). This is the difference between observing, for example, a book on a desk and remembering that the book is on the desk, as opposed to remembering being told the sentence “the book is on the desk.”

Successful use of the GALIS framework to perform a card matching task, and its previous use to perform simpler n -Back matching tasks in the preceding

chapter, is very encouraging. However, much further work is needed to assess this approach and extend it to even more challenging problems. Since neither sophisticated image processing or motor control were a focus of this work, expanding such portions of the system to, for example, deal with color and invariance to input transformations, would of course be important future research areas. In addition, “programming” a neural network as we have done here is a fairly new pursuit, and one that would benefit from finding new methods and tools for analyzing the behaviors of large-scale complex network architectures.

Learning, Improvement & Task Switching

6.1 INTRODUCTION

The GALIS model from the previous chapter demonstrates the approach's ability to co-process visuospatial information and use it to execute a complex task. This was done using an algorithm which is stored as a set of instructions in one of its memory layers. One of the primary limitations to its performance was this stored program itself, rather than the network per se. This limitation is addressed here with a network capable of improving on its performance and learning to adjust the representation of its own instructions during execution.

The goal of the work described in this chapter is to allow GALIS models to learn procedural knowledge from their experiences and to improve on tasks from exposure to them. Sub-symbolic systems are typically fairly adept at improving from experience via incremental weight changes. While GALIS does learn to store information—e.g., its working memory, as well as learning its initial instructions—the GALIS models in the preceding chapters do not adapt these instructions based on their experience during task execution. GALIS models so far have been limited to being only as good as the instruction set

their creator provided them with. The work in this chapter overcomes this limitation by combining the rapid, one-shot learning used thus far in GALIS for external episodic memories and instruction sequences with the gradual improvement more typical of neural networks. Here this capability is referred to as *instruction refinement* to differentiate it from the instruction learning that has been discussed since Chapter 4.

In addition to instruction refinement, the ability of GALIS models to bind “what” and “where” information from their environments is expanded upon to now also include the binding of multiple features — such as number, color and shape — concurrently. The model must attend to one of these features while inhibiting the others, which presents a more complex cognitive control problem than attempted in the previous chapters. The model of the previous chapter could recognize having seen a ‘X’ image on a card, but here the model must remember that it saw an image with three red crosses on it, and must also be able to remember where else it has seen cards with red shapes of any kind, or crosses of any color.

In order to facilitate rapidly switching between which of these features is attended to and which inhibited, there is a third variation on instruction learning used here. Instruction refinement makes small, permanent changes to the controller’s weights, while more drastic but temporary changes are needed to shift the network into a new set of behaviors in response to negative feedback from the environment. These three types of learning used for instructions in this GALIS model are summarized in Table 6.1.

The chapter begins with some further background and motivation of the problem. In order to test GALIS’ ability to improve with experience, a model is built which performs the Wisconsin Card Sorting Test (WCST). This test and

TABLE 6.1. Types of learning used in the GALIS WCST model.

	Instruction learning	Instruction refinement	Rule set switching
Purpose:	initial memory contents	iterative improvement	rule changes
Speed:	very fast	slow	fast
Method:	1-shot Hebbian (symmetric & asymmetric)	bounded Hebbian & bounded anti-Hebbian	bounded Hebbian
Weights:	$\mathbf{W}_{\text{ENC}}, \mathbf{W}_{\text{DEC}},$ $\mathbf{W}_{\text{ISM}}, \mathbf{V}_{\text{ISM}}$	\mathbf{W}_{ISM}	$\mathbf{U}_{\text{ENC}}, \mathbf{U}_{\text{ISM}}$





the methods used in the model which executes it are described in the following section. Finally, results on the WCST are presented which show that GALIS is not only capable of performing the WCST, but also of improving over time, and in learning to distinguish between useful instruction sequences for the task and spurious ones.

6.2 METHODS

6.2.1 THE WISCONSIN CARD SORT TEST

The WCST is one of the primary tests of cognitive control (Greve et al., 2005; Strauss et al., 2006). It is widely used to study executive functions and the pre-frontal lobes (Barceló and Knight, 2002), for example in studies of schizophrenia (Everett et al., 2001) and ADHD (Romine et al., 2004). It comes in various forms (Berg, 1948; Milner, 1963; Nelson, 1976), but all are performed using a deck of cards whose images differ in three dimensions, each of these which has four possible values. (In this implementation, these are shape, color and number, as is standard; see Table 6.2.) The test is conducted by placing four “base” cards in a row, and presenting the participant with a series of cards

TABLE 6.2. WCST input features and values.

number		color		shape	
1		red	rgb(1,0,0)	square	■
2		blue	rgb(0,0,1)	cross	+
3		cyan	rgb(0,1,1)	triangle	▲
4		yellow	rgb(1,1,0)	circle	●

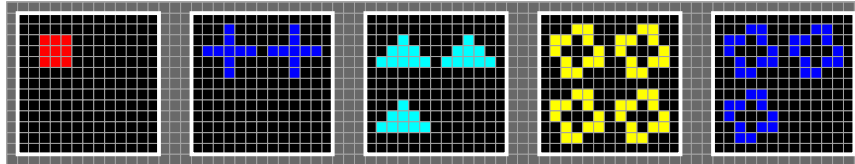


FIGURE 6.1. A sample visual field for the WCST. The four cards depicted on the left are basis cards showing one red square, two blue crosses, three cyan and four yellow circles. The stimulus card on the far right depicts three blue circles, and could match the 2nd, 3rd or 4th bases depending on whether the active dimension is color, number or shape, respectively. (Best viewed in color.)

from the deck, one at a time. The participant's task is to match the current stimulus card to one of the four base cards, but they are not told what rule should be used for the matching. Figure 6.1 depicts the low-resolution version of the WCST that the GALIS model sees. Each stimulus card could match with three of the four basis cards depending on the currently active dimension. The active dimension must be attended to while the other two must be inhibited. Participants are not even informed that there are three potential rules: they may believe they should be matching based on most overall similarity, or any more complicated rule of their devising such as choosing the card with the same shape if the stimulus is red, but the same number if it is any other color.

During the course of the test the relevant dimension will change without the participant being informed. Depending on the version of the test, this

switch usually occurs after a prescribed number of consecutive correct responses (typically six to ten). The most common measure of performance is the number of rules shifts or “sets” one is able to complete before the deck is emptied.

The only feedback participants receive is “correct” or “incorrect” following their response to each new stimulus. Using only this binary reward/punishment signal, the agent must determine whether they are attending to the correct feature. This creates two potential errors: either continuing to attempt to apply the same rule after it is no longer working (called “perseveration”), and switching away from a rule while it is still in effect (called “distractibility” or “failure to maintain set”). The WCST is often referred to as a test of set shifting ability, or the ability to switch from one mental frame to another, changing the features of the environment which are being attended to and which inhibited (Boone, 1999). One of the principal challenges of cognitive control is to work with mental constructs which are flexible enough to adapt quickly while simultaneously remaining robust to unwanted change. This is evident in the WCST, where a failure to meet the first criteria results in *perseverative error*, and a failure to meet the second results in a *distraction error* (Stemme et al., 2007).

6.2.2 MODEL OVERVIEW

The use of the WCST as a testbed for this stage of GALIS’ development is quite a challenge. The principal demand of WCST is to be able maintain a stable focus on one feature dimension while also being able to rapidly switch between features when necessary. This requirement is made more difficult by the addition of gradual improvement, creating a trichotomy between not changing behavior, making a large change quickly, and making many small

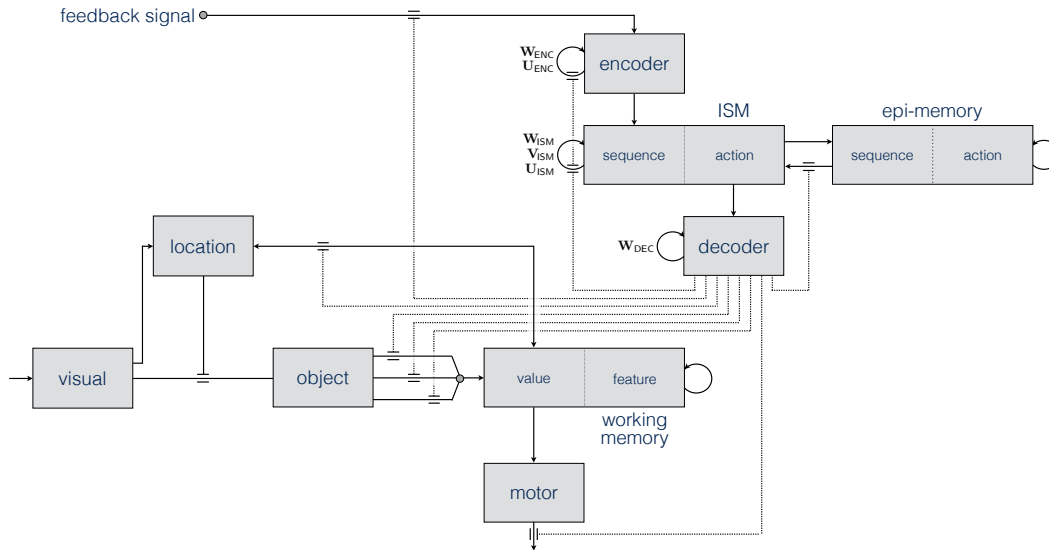


FIGURE 6.2. Model architecture for the Wisconsin Card Sort Test. Solid lines represent the direct flow of activity between layers. Dotted lines represent gates. The names of the Control Module’s weight matrices appear next to the layer they operate on.

changes slowly. All of this must be accomplished on the basis of nothing but a binary reinforcement signal from the environment.

The GALIS model for performing the WCST is composed of seven modules, as illustrated in Figure 6.2. These are the Visual, Location, Object, Motor, Working Memory, and Controller modules. Many of these components are extremely similar to those used in previous GALIS models; further information can be found in earlier chapters as well as Sylvester et al. (2013) and Sylvester and Reggia (2014). The remainder of this section will concentrate on the differences between this and those previous GALIS models.

6.2.3 INFORMATION REPRESENTATION

Like the networks presented in the previous chapters, this model uses randomly selected binary strings to represent different concepts in a distributed fashion.

Each sequence of instructions has an identifier, as do the constituent actions within those sequences. Each feature is represented by such a string, as well as the particular values they can take. The set of these strings is referred to as the model’s “codebook.” These are the patterns which are learned by the model’s attractor networks, and the points in state-space corresponding to these strings become the centers of the basins of attraction.

In prior chapters these strings were chosen randomly. As a result some were more similar than others, resulting in deformations to the evenness and uniformity of the attractor space. In order to ameliorate this issue, we instead choose encodings using an iterative sampling process (Kulesza and Taskar, 2010, 2012; Usatenko et al., 2014). When new encodings are generated they are rejected with some probability based on their distance from the already adopted encodings. That is, the first entry in the “code book” is chosen at random. The second is chosen, but is likely to be rejected if it is too close to the first.¹ The process is repeated until the codebook is full. While not algorithmically elegant, it is more than sufficient because the number of points being chosen is much, much smaller than the dimensionality of the space they are in (Pan et al., 2007). The strings which identify features and their values are 512 bits long. Those for sequences are 600 bits and those for actions 424 bits, matching the sizes of the appropriate areas of the ISM.

¹ Its likelihood of being rejected actually increases if it is too close to the prior selections *or too far*, i.e., too close to a prior selection’s complement. This is because auto-associative memories form spurious attractors around the complements of the patterns they are trained on. Each attractor should be far away from the others, and also far from the unwanted doppelgängers that they entail.

6.2.4 VISUAL SYSTEM

The Visual System comprises the Primary Visual, Location and Object layers. The Primary Visual layer provides the principal input to the network. This comes in the form of a visual field of 15×80 pixels. (Figure 6.1.) Each pixel is represented by three nodes: one each for red, green and blue intensity. Cards are 13×13 pixels, with three colors per pixel, each having a real value in $[0,1]$.

As in the preceding chapter, the GALIS Visual System is inspired by the dual pathways of the mammalian brain, which connect the visual cortex to the prefrontal cortex along parallel pathways. The first of these is the parietal or dorsal pathway, which focuses on information about where objects are, and the second is the temporal or ventral pathway, which specializes in what objects are (Ungerleider and Haxby, 1994). Within this GALIS model, these pathways are represented by the two layers to which the Primary Visual layer sends its output: the Location and Object layers.

The Location layer is responsible for judging where a stimulus is being observed, but not for any details about its appearance. The Object layer is responsible for more fine-grained observation of stimuli. It provides details about the appearance of objects, but is capable of processing only a small region at a time, which results in it being ignorant about where the stimulus it is attending to is located. The information from both of these must be reintegrated by the Executive System.

The Location layer combines bottom-up with top-down influences to determine where in the visual field the model should attend. It directs the model's attention to a particular location by gating the connection between the Primary Visual and Object layers: only those connections between the two that link the

attended-to location are opened. As this system is identical to that described in the previous chapter it will not be discussed further here.

Though the Location layer is identical to that of Chapter 5 and Sylvester and Reggia (2014), the Object layer has been greatly modified to allow it to do feature-extraction before communicating with the Executive System. To accomplish this the Object module now consists of two layers: a Self-Organizing Map (Kohonen, 1998) which learns to extract features from the visual scene, and an output layer which forms a consensus of the nodes of SOM. The SOM is trained prior to executing any WCST trial. (In essence we assume that the subject of a WCST trial is already capable of discriminating between squares and triangles, between yellow and blue, etc.) A 40×40 rectangular grid of nodes is trained using the standard SOM learning algorithm with all cards in the deck as inputs for 3000 epochs. A subset of a trained SOM is depicted in Figure 6.3. The nodes of the SOM output to a second layer of 1536 nodes (512 each for shape, color and number). This layer provides the ultimate output consisting of the Object module's opinion of the value of all three features of the current visual input. The Object module is agnostic about which matching rule is currently in effect — indeed, it is unaware that any matching is even going on. It merely passes its determination of shape, color and number upstream to the controller to do with as it will.

The weights w_{ij} from a node i in the Visual layer to a node j in the SOM are trained according to the standard competitive learning SOM algorithm mentioned above. This results in a SOM node's activation being determined by $a_j = \sum_i w_{ij}s_i$, with s_i being the state of node i and a_j that of node j . The weights from nodes in the SOM to the Object output layer are determined analytically based on which features the SOM node responds most strongly to.

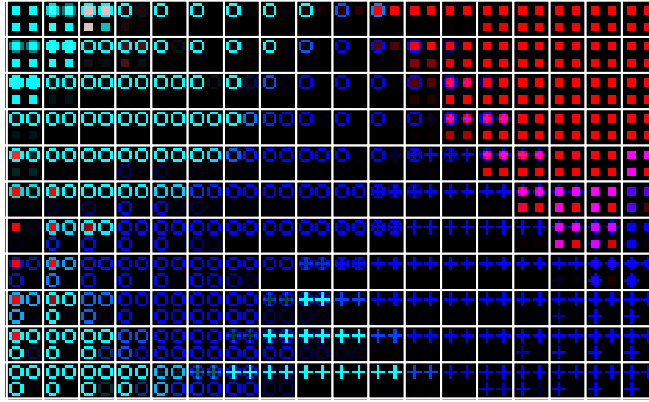


FIGURE 6.3. Detail from an example Object layer SOM, with each node labeled with what the card that would make it maximally active would look like. For example, the node in the upper left corner has learned to represent four cyan squares, while that in the lower right corner has learned to represent three blue crosses. (Best viewed in color.)

The procedure is more straightforward with an example: if a node responds more strongly to red more than any other color, squares than any other shape, and two items more than any other number, its outgoing weights will be set as the distributed encodings chosen as in §6.2.3 for red, square and two. Formally, this can be expressed as $\mathbf{w}_j = \mathbf{c}_p$ such that

$$p = \arg \max_{q \in \{1,2,3,4\}} \left(\sum_{\lambda \in S_q} \sum_i w_{ij} s_i^\lambda \right) \quad (6.1)$$

Here \mathbf{c}_p is the one of the four entries in the codebook for a feature-value, and S_q is the set of all cards in the deck with that value. (In the case of color, we might have c_1 be the 600-bit code for red, c_2 be the code for blue, etc., and thus S_1 is all the red cards, S_2 all the blue cards, etc.) The rightmost summation above is the activation of SOM node j in response to all of the pixels in card λ . The left summation then yields node i 's activity as a result of all—for

example — red cards. We set the desired weight to be the codebook entry \mathbf{c}_p for whichever feature-value causes j to be maximally active.

The ultimate output of the Object layer is a summation of the outputs of each node in the SOM, weighted by their activity in response to the current visual input. For an Object output k then, this is

$$o_k = \sigma \left(\sum_j w_{jk} a_j \right) \quad (6.2)$$

where $\sigma(\cdot)$ is the logistic sigmoid function. This process gives single, consensus view of the value of the visual input for each feature. These three values are later combined based on the state of the three visual gates to create a single 512-bit value which is passed to the Executive System. Once again, an example may help clarify: if the only nodes in the SOM with non-zero activation respond most to **two red squares** and **two red circles** then the final output will be the distributed encoding for **two**, that for **red**, and an affine combination of the encodings for **square** and **circle**.

6.2.5 WORKING MEMORY

The WCST is a game of full information with respect to the visual environment. All four base cards and the stimulus cards are in view at all times. The memory which is required is that of the agent’s own recent actions and the feedback to them. As a result, the Working Memory layer is pre-trained with the relevant attractors. A new component called the “Epi-Memory” is used by the model to form a memory of its own recent actions, discussed below in §6.2.6.

The role of the Working Memory is to link features to locations. The card matching model linked “what” and “where” strings together in a single auto associative memory in the same way that the ISM links sequence and

action identifiers together. Here the Working Memory of the card matching model is expanded to link not just *what* with *where*, but *number*, *color* and *shape* with *where*. To accomplish this the network is divided into two sections conceptually (i.e., the network is fully connected but we interpret the two halves as representing two different things), just as was done for the what and where portions when modeling card matching. The first portion identifies a feature dimension — number, color, shape and location — while the second identifies a particular value of one of these features. Using one-shot Hebbian learning sixteen attractors are created in the network, one for each feature-value pair. Then these attractor states are linked together using asymmetric Hebbian learning. For example, given the board depicted in Figure 6.1, the attractor for (color; red) would be linked to the attractor representing (location; position #1), as would those for (shape; square) and (number; one). This ‘linking’ is identical to that discussed in prior chapters to connect one element in a sequence to the succeeding element. Until now this technique has been used to link, for example, an observation at time $t - 1$ to an observation at time t , but there is no reason that the heteroassociative bond between these two patterns can only be formed when patterns are *temporally* related. In total, there are 16 attractors learned using the symmetric weights of \mathbf{W}_{WM} , which are linked together into 12 sequences of 2 elements each using the asymmetric weights of \mathbf{V}_{WM} (Figure 6.4). These two weight matrices used within the Working Memory layer are equivalent to the symmetric and asymmetric weights introduced and then refined in Chapters 3 and 4.

Due to the ability of an associative memory to recall full patterns from partial ones, it is possible to recover a full state such as (color; red) given only the input red from the Visual system. Using the asymmetric weights, a further

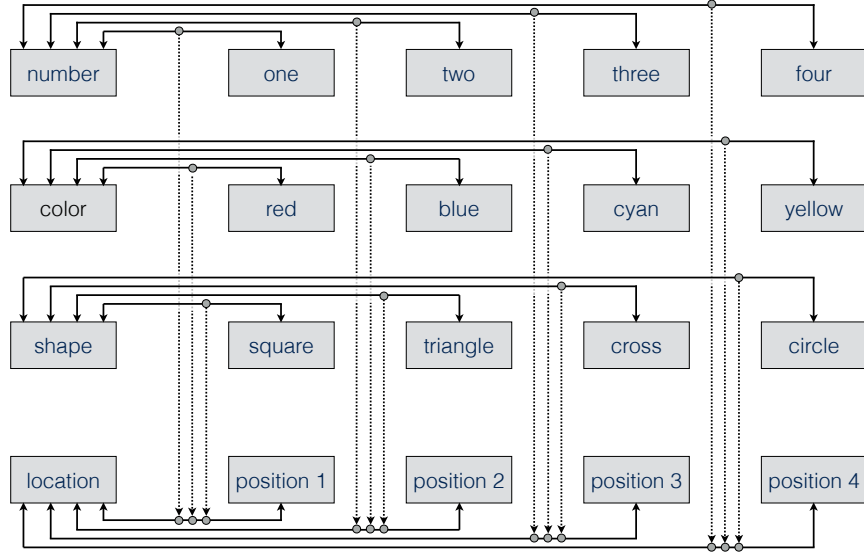


FIGURE 6.4. A schematic of the associations between feature-values and locations. Solid, horizontal lines show associations between a dimension and one of its possible values that have been made using the symmetric weights of \mathbf{W}_{WM} . Dashed, vertical lines show the associations that have been formed between the resulting attractors using the asymmetric weights of \mathbf{V}_{WM} .

update of the network will then be able to recall the representation for (location; position #1), and this value can be passed to the Motor Layer to allow it to gesture at the basis card in the first (leftmost) position (see Figure 6.1).

6.2.6 CONTROLLER

The Controller is responsible for storing the instruction sequences the model uses. It is composed of the same three components as the previously presented GALIS models — Encoder, ISM and Decoder — with the addition of one more component to enable instruction refinement and rule switching, which is termed the “Epi-Memory.” This section will concentrate on the new Epi-Memory component, and the reader is referred to Chapters 4 & 5 for details on the remainder of the controller, as these have not changed.

The Epi-Memory is used to keep a record of the actions recently taken by the Controller. Rather than keeping a record of the sequence of visual stimuli, as for instance the n -Back model's Working Memory layer did, the Epi-Memory records the internal state of the ISM. It is the same size as the ISM (1024 nodes), and forms a sequential memory using the same attractor network techniques as the other memories in the model: asymmetric weights, dynamic thresholds, and weight decay (See Chapter 3). When the model receives feedback this memory is then used to retrieve the recent actions taken and either strengthen their representations in the Controller — in the case of positive feedback — or weaken them — in the case of negative feedback.

6.2.6.1 GATES & INSTRUCTION SET

The GALIS model for the WCST is governed by the following gates (see Figure 6.2).

- Three dimension gates, one each that controls the relative contribution of shape, color and number encodings from the Object layer. If all are closed, the Executive system will not “see” anything; if just the shape gate is open it will only perceive the shape of the object being viewed, etc.
- A motor output gate, that when open allows the Motor layer to gesture to the basis card that the model has chosen as match to the current stimulus.
- An encoder update gate as in previous models.
- A gate to control the top-down attention in the Location layer, as in Chapter 5.

TABLE 6.3. Instruction sequences stored by the control module.

Sequence	Action
1. Attend to number	1. process feedback & update SOM
	2. open number gate
	3. retrieve location from WM
	4. output
2. Attend to color	1. process feedback
	5. open color gate
	3. retrieve location from WM
	4. output
3. Attend to shape	1. process feedback
	6. open shape gate
	3. retrieve location from WM
	4. output

- A gate on the reinforcement signal that opens when the model the ready to accept process feedback.
- A gate to govern the updating of weights in the Controller, in the same way that previous models had gates to update weights in the Working Memory.

A gloss of the instruction sequences used, similar to those in previous chapters, is given in Table 6.3. There are three similar sequences, differing only in which of the visual gates is opened and therefore which feature dimension the model will be using to determine a match. This makes the algorithm used simple to understand but difficult for the model since there is a high degree of overlap between the sequences.

The first step is to process the feedback signal from the environment in the wake of the model’s previous answer. If the feedback gate is opened the internal weights of the Control Module will be updated as described in the next

section. Next, one of the three visual gates is opened, which allows the Object’s layer’s decision on the value of the corresponding feature to be sent as input to the Working Memory. Next the working memory is updated which causes the location of the basis card that matches the stimulus card in the chosen dimension to be retrieved. Finally this location is output to the environment and the cycle repeats.

6.2.6.2 NEW VARIETIES OF INSTRUCTION LEARNING

There are two new facets of learning being used here. The first is *rule shifting*, which is a temporary change in the controller’s weights to enable it to change which instruction sequence it is executing in response to negative feedback. The second is *instruction refinement*, which involves repeated, marginal changes to the controller’s weights to produce increasing performance over time.

Rule Shifting. In order to enable shifting between attentional sets and improving with experience, two new weight matrices have been added to the controller: \mathbf{U}_{ISM} and \mathbf{U}_{ENC} . These act as “fast weights” to change the dynamics of the ISM and encoder, respectively. By dividing the responsibility between the fast and standard/slow weight matrices it is possible to make rapid but temporary changes to the behavior of a network without affecting its behavior in the long term (Gomez and Schmidhuber, 2005; Hinton and Plaut, 1987; Reggia and Edwards, 1990; Schmidhuber, 1992; Tieleman and Hinton, 2009). For instance, if a network has learned to store a set of patterns using Hebbian learning on matrix \mathbf{W} , then one could temporarily remove a pattern from the network’s memory by doing Hebbian learning of that pattern on a matrix \mathbf{U} and using $\mathbf{W} - \mathbf{U}$ as the connection weights when updating the network. By setting \mathbf{U}

back to zero the network can be returned to the state it was in at the outset without affecting its memory of the pattern in question or any others.

Bounded Hebbian learning is used to train the fast weights (Gerstner and Kistler, 2002). This scales the magnitude of weight changes to prevent them from overwhelming the existing weights. Following positive feedback, recent ISM states are retrieved from the Epi-Memory just as recent stimuli were retrieved when carrying out n -Back. A recent state x is then used to update \mathbf{U}_{ISM} according to:

$$u_{ij}^{\text{ISM}} = (1 - k_{\text{U}}) u_{ij}^{\text{ISM}} + \frac{1}{n} \phi(\psi - |u_{ij}^{\text{ISM}}|) (x_i x_j - \delta_{ij}) \quad (6.3)$$

where k_{U} is decay rate of \mathbf{U} and ϕ and ψ are parameters to control the boundedness of the weight updates.²

Like the n -Back and Card Matching models, the ISM's state is updated twice per time step: once using the asymmetric weights \mathbf{V}_{ISM} to move the state to a new attractor, and then again using the symmetric weights \mathbf{W}_{ISM} to settle the state more fully in the new attractor basin. When the ISM is updated according to its input equations

$$\mathbf{a}_{\text{ISM}} = \text{sgn}[\mathbf{V}_{\text{ISM}} \cdot \mathbf{a}_{\text{ISM}} - \boldsymbol{\theta}_{\text{ISM}}] \quad (6.4)$$

$$\mathbf{a}_{\text{ISM}} = \text{sgn}[(\mathbf{W}_{\text{ISM}} + \mathbf{U}_{\text{ISM}}) \cdot \mathbf{a}_{\text{ISM}} - \boldsymbol{\theta}_{\text{ISM}}] \quad (6.5)$$

it will be more likely to enter states which have recently received positive feedback, which thereby makes distraction errors less common. The supplementary, bounded learning works when the model is behaving correctly because it deepens the basins the network is already in. The intuitive explanation is

² The experiments described here use $\phi = 0.3$ and $\psi = 0.2$, but the results are not particularly sensitive to the specific values, especially if the product $\phi \cdot \psi$ is small.

“what you just did worked; do it more in the future,” or more tersely “don’t shift rules now.” In a changing environment like that of the WCST what has worked in the recent past is no guarantee of future success, hence these changes being made to the decaying, temporary \mathbf{U}_{ISM} and \mathbf{U}_{ENC} .

What is needed after incorrect responses is to switch rules by moving to a different basin. To do this anti-Hebbian learning is used in \mathbf{U}_{ENC} to temporarily remove the sequence which provided the incorrect answer. This is done by

$$u_{ij}^{\text{ENC}} = (1 - k_{\text{U}}) u_{ij}^{\text{ENC}} - \frac{1}{n} (x_i x_j - \delta_{ij}) \quad (6.6)$$

Since $|u_{ij}^{\text{ENC}}|$ is typically negligibly small, the above can be viewed as the same bounded Hebbian learning process as Equation 6.3, but with bounding parameters $\phi = -1$ and $\psi = 1$. The fast weights for the encoder are combined with that network’s standard weights during state update, making it unlikely to remain in a sequence of actions which is yielding negative feedback.

$$\mathbf{a}_{\text{ENC}} = \text{sgn} [(\mathbf{W}_{\text{ENC}} + \mathbf{U}_{\text{ENC}}) \cdot \mathbf{a}_{\text{ENC}}] \quad (6.7)$$

Note that the Card Matching model of the previous chapter accomplished the temporary suppression of a state using activity from the Register layer to provide countervailing biasing inputs. The idea was that this would “push” the network out of one attractor basin and thereby allow it to enter another. This was largely effective, but occasionally this resulted in moving the network out of the current attractor but into a spurious attractor that is the complement of the intended, trained pattern. Using fast weight matrices to temporarily unlearn the attractor state in question is a more effective method since it does not require that the biasing influence be so precisely matched against the influence of the network’s weights.

Instruction Refinement. The system for long-term improvement via instruction refinement is very similar to that for set-shifting. In the case of positive feedback, a bounded Hebbian learning rule is applied to \mathbf{W}_{ISM} and \mathbf{W}_{ENC} to enlarge basins which resulted in reward. For \mathbf{W}_{ISM} this rule is given by

$$w_{ij}^{\text{ISM}} = w_{ij}^{\text{ISM}} + \frac{1}{n} \phi (\psi - |w_{ij}^{\text{ISM}}|) (x_i x_j - \delta_{ij}) \quad (6.8)$$

and the same is used, *mutatis mutandis*, for \mathbf{W}_{ENC} . Because these modifications to the model’s instructions are permanent they are made more gradually, so smaller values of ϕ and ψ are used than with the changes to \mathbf{U}_{ISM} described in Equation 6.3.³ Following negative feedback the same system is used on \mathbf{W}_{ISM} and \mathbf{W}_{ENC} , but with a negative learning rate to weaken the associated attractors instead of strengthen them (*cf.* Tieleman and Hinton, 2009). By repeated marginal strengthening of the attractors which result in positive feedback and weakening of those which result in negative feedback the network is able to fine-tune its representation of its instructions to support increasing performance levels as time passes.

6.3 RESULTS

Experiments were initially run using a reduced deck of cards, in which only those cards which differ from the basis cards in exactly one dimension appear (Dehaene and Changeux, 1991; Milner, 1963). Those cards that are in the deck appear twice, giving a total of 48 cards. The correct dimension was switched after 6 consecutive correct responses. Tests using the full deck and a double deck—

³ The work here uses $\phi = 0.03$ and $\psi = 0.01$. Once again the particulars of this choice are not significant, but it is important that the product of these two parameters be sufficiently small.

both common versions of the test — were also run. These did not produce any meaningfully different results but did require significantly more computational time simply due to the increased number of cards to be processed per trial, so I present results based on reduced deck tests.⁴

Models were run on the test from the beginning 12 times consecutively. Between rounds all the layers of the controller had their states reset to random values, and all fast weights \mathbf{U} and dynamic thresholds θ were reset to zero. (Note that the changes made to \mathbf{W}_{ISM} and \mathbf{W}_{ENC} via instruction refinement were *not* reset or undone.) A total of 100 trials of 12 consecutive tests each were run for models with instruction refinement activated and without. The mean number of sets completed by the models are shown in Figure 6.5. Without epi-learning, there is no improvement from the first to the twelfth trials. With epi-learning, however, there is improvement as the model gains experience. This improvement is significant at the $p < 0.05$ level as judged by a Kruskal-Wallis test. While the improvement in performance may appear modest, the number of sets completed increased from 3.86 to 4.97, a 29% increase.

Figure 6.6 shows a histogram of the number of sets completed by the instruction refinement model in the first test compared to the last. The shift in performance is evident: the number of trials which completed four sets or less decreases while the number completing five or more increases. It is also worth noting that variation in the number of sets completed across trials decreased steadily as the models gained experience, indicating a less erratic behavior as time went on. The standard deviation was 1.02 for the first round both with

⁴ For a discussion of the applicability of the reduced test in comparison to the full version in human subjects, see Smith-Seemiller et al. (2001); the authors report on meta-analysis that concludes short form scores are highly correlated with those on the long form, though less so for very young or old subjects.

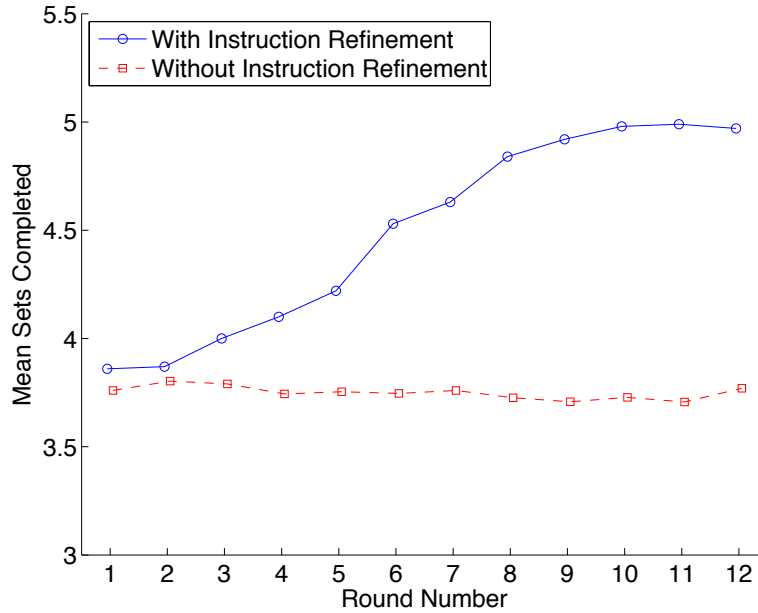


FIGURE 6.5. Mean number of 6-card sets completed by GALIS models both with and without instruction refinement capability, averaged over 100 trials. (Note that the vertical axis does not begin at 0.0.)

and without instruction refinement; without instruction refinement, this value remained roughly constant at 0.98, but with instruction refinement it decreased to 0.76 after 12 rounds.

In addition to testing the model’s ability to refine its representation of the three “correct” instruction sequences it was given, I also ran experiments to judge the GALIS model’s ability to distinguish between useful and not useful instruction sequences. This was done by training the model on an additional sequence which was similar to the three given in Table 6.3, but instead of opening one of the gates from the Object layer to attend to a particular feature, all three were opened one third of the way. (Note that human subjects are not told they should be matching based on only one feature at a time. The rule “select the basis card with the highest overall similarity to the stimulus” is a

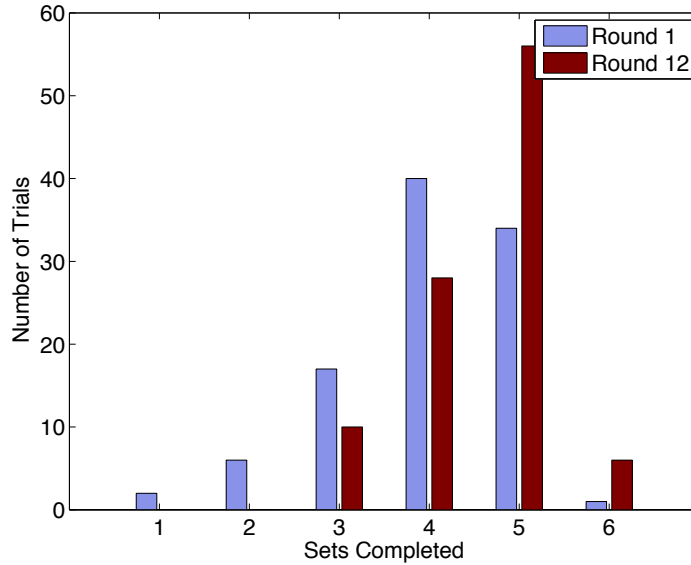


FIGURE 6.6. Distribution of the number of sets completed by networks with instruction refinement on their first test and their twelfth. Experience reduces the number of poor performing trials and increases the number of high performing trials.

perfectly valid option, despite never being correct. This is the rule which the new, fourth instruction sequence encodes.)

The addition of this supernumerary sequence resulted in an initial drop in performance. This should not be surprising: not only does the model now have an extra, incorrect option to choose from during every rule shift (Dehaene and Changeux, 1991), but there is the added problem of a more crowded attractor space. Performance thus drops for both algorithmic and neural reasons. Figure 6.7 shows the average improvement from this initial state over the course of 15 games: by the end of this period the networks have improved from an initial performance of 2.8 sets completed per game to 3.7. This final performance level was not significantly different from the initial performance when only the three correct instructions were trained, presumably indicating that the model successfully learned to ignore the unnecessary sequence.

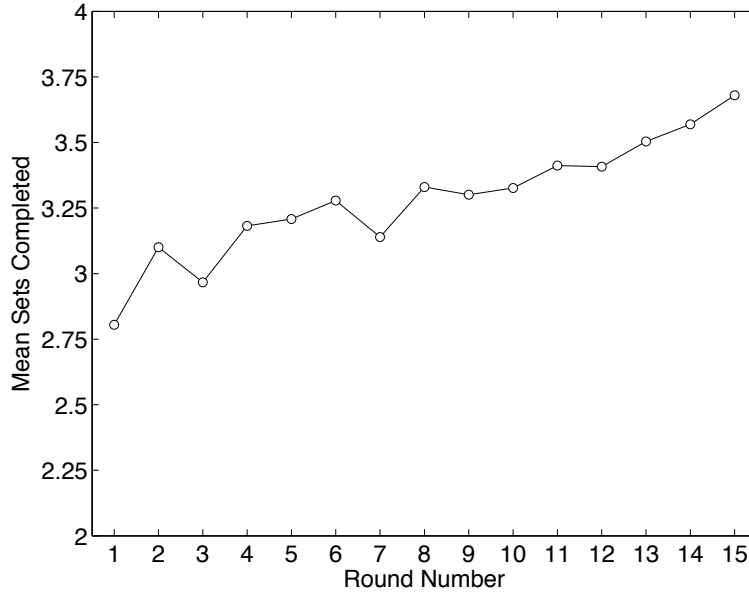


FIGURE 6.7. Mean number of 6-card sets completed by GALIS model over 100 trials, when the controller has been trained on the three correct instruction sequences listed in Table 6.3 as well as one extraneous sequence. (Note that the vertical axis does not begin at 0.0.)

To further explore this adaptation, I also tracked how often each of the four sequences was selected by the control module over the course of a test. Figure 6.8 shows the number of times each was selected during the first and fifteenth games in 100 trials. Though the extraneous instruction sequence was selected less often than the three correct sequences in the first game of each trial, this difference was not statistically significant. By the fifteenth trial, all three of the correct sequences have increased their likelihood of being acted upon. This comes at the expense of the fourth sequences, which occurs only two thirds as often as it did before the model gained experience. It is now significantly less likely to be selected by the model than the other three sequences, showing that the model learned to differentiate between useful and useless instructions.

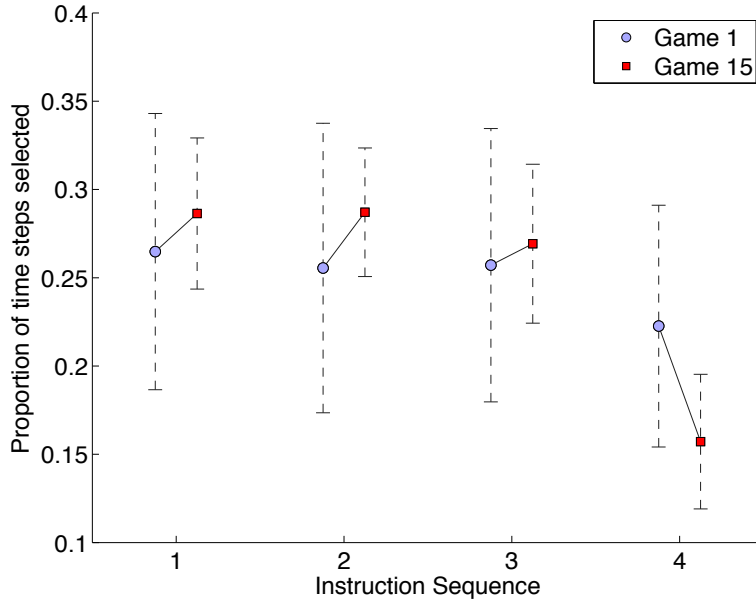


FIGURE 6.8. The proportion of time steps each of the four instruction sequences (#1–3 correct; #4 superfluous) were selected by the controller, averaged over 100 trials. From game 1, when the model has no experience, to game 15 the likelihood of executing all three of the correct instruction sequences increases, while the likelihood of selecting the superfluous sequence #4 decreases. (Note that the vertical axis does not begin at 0.0.)

6.4 DISCUSSION

Here we have demonstrated that GALIS can not only store the instructions for a task in its memory, but can also autonomously adapt the contents of that memory to allow better performance. One of the benefits of the GALIS approach—the ability to “program” a neural network—can also be viewed as a potential weakness. To wit, these programs must be determined *a priori* by the modeler rather than deduced by the network. While this GALIS network does not derive its instructions itself, it does collaborate with the modeler in improving upon the instructions it is given. It does so by reinforcing the

attractors for instructions that it has experienced as being useful and dampening those not found to be so. This is important for several reasons. Firstly, GALIS models can be susceptible to choices of patterns which result in attractor basins being too close together. Instruction refinement in the controller makes the network more robust to this situation by strengthening attractors which result in positive performance. Secondly, the instructions do not need to be finely-tuned, only sketched out because the model itself is capable of performing this tuning operation. Thirdly, because the model can learn to ignore irrelevant patterns, the modeler has the flexibility to offer instructions which may be useful, and have the network discover for itself which of them actually are.

This self-directed improvement was possible within the bounds of the GALIS framework, using regions of attractor memories, linked with gating. The fast-weights concept used in rule switching is closely tied to gating, as discussed in Chapter 5. Both are strategies which allow one network to influence the behaviors of others. The two concepts are largely congruent; you could describe all of the opening and closing of gates as the modification of (very) fast weights on those gated connections and vice versa.

It was also possible in the context of a demanding cognitive control task which requires careful balance between stability and plasticity. In addition to incorporating the abilities to not change (stability) and change quickly (plasticity), the work here also adds the ability to change *slowly* (instruction refinement).

Incremental improvement is one of the areas that neural networks often have an advantage in compared to their symbolic counterparts. By introducing this previously-lacking ability to GALIS it has become a stronger potential link between the two paradigms.

Alternatives to bounded Hebbian learning should be explored further. Initial explorations of other options, including Storkey learning rules (Hu, 2013; Storkey, 1997; Storkey and Valabregue, 1999), were inconclusive so it was decided to remain with Hebbian-based rules to provide continuity with the rest of the model. Nevertheless, more rigorous testing of alternatives may be fruitful. Further work should also be done on GALIS learning to ignore irrelevant instruction sequences. Currently this is only possible for a small number of such supernumerary sequences as any more cause the attractor space to be too crowded to perform at a high enough level to receive positive feedback, which is a requirement for undergoing the instruction refinement process. Other memory paradigms with higher capacities, or indeed the same paradigm but with a larger computational substrate, may allow a larger set of instruction sequences to be learned as candidates. Other possibilities include introducing new sequences as old ones decay away, or initializing different networks with different candidate instructions and monitoring their performance across agents.

Discussion

This chapter concludes this dissertation by summarizing the work done on neural models of executive behavior and highlighting the original contributions made to the field. It also covers the limitations of this work in addition to possible future directions for it.

7.1 SIGNIFICANCE AND SUMMARY

The fundamental issue addressed in this work is whether there is an identifiable core set of of general-purpose, region-level functions and interactions that can be used for cognitive control in large-scale neurocognitive architectures. The hypothesis is that the GALIS framework provides such a set of three key functions and interactions: a region-and-pathway architecture inspired by the organization of the human cerebral cortex and biologically-plausible hebbian learning, neural regions that each serve as an attractor network that is able to learn temporal sequences, and neural regions that not only learn to exchange information but also learn to turn on/off the functions of other regions. The idea of simulated cortical regions that can gate one-another's activations, learning and communications is particularly novel.

Most machine intelligence systems fall into one of two general groups: systems that take a symbolic, top-down approach, and those that adopt a neural, bottom-up approach. The divide between these two strategies is both long-standing and, at times, quite contentious. This is regrettable because the two different strategies are in many ways complementary rather than competitive: each of the two approaches has its own relative strengths and weaknesses. For example, while neural systems excel at problems that involve pattern matching, incremental learning, low level control, fault tolerance, or processing noisy data, they are less adept at handling higher cognitive functions such as goal-directed reasoning, natural language processing, meta-cognition, and planning. Top-down symbolic methods are largely antipodal.

In the cognitive modeling domain, the current limited abilities of neural architectures to capture critical aspects of high-level cognition puts them at a tremendous disadvantage when trying to model the processes underlying human cognitive control. This limited ability of neurocomputational methods to support high-level cognition is somewhat unexpected in that the human brain handles such issues routinely, establishing that neural computations clearly have the capacity to do so. It also hampers progress in understanding intelligence, as we are unable to connect our knowledge and experience of intelligent behavior at the macro level with our vast and growing body of information about the operation of the brain at the micro level (Reggia et al., 2014). I believe that a bridge between symbolic and neural approaches will be very advantageous, and that the GALIS framework is one way to advance this reconciliation. To the extent that it and other related research is successful it may even contribute to a better understanding of the general mind-brain problem (*ibid.*).

The complementarity between the symbolic and neural paradigms has been recognized in the past and leveraged effectively in a number of cognitive architectures (e.g., Sun and Naveh, 2004). However, attempts to graft one approach on to the other have been largely unsuccessful (e.g., Jilk et al., 2008). Conversely, more recent attempts to synthesize the two have been meeting with growing success and interest (Beck et al., 2008; Dayan, 2007; Holyoak and Hummel, 2000; Stewart et al., 2011). In many ways this mirrors the recent shift in other domains to hybrid discrete-continuous systems, such as vector space methods in natural language processing and information retrieval (Le, 2012; Smolensky et al., 2014; Socher et al., 2012)

This dissertation focuses on a potential avenue to attempt to bridge this gap through the creation of neural networks with memories not only of their external environment but of internal actions and procedures. GALIS makes use of multiple interacting networks, many of which are attractor-based memories, and which influence each other's operation through the use of gated connections. This results in biologically plausible neural networks which nevertheless exhibit behaviors typical of both symbolic and sub-symbolic approaches. The use of distributed representations, high dimensional attractor spaces, non-linear interactions between layers and one-shot Hebbian learning are all characteristic of neural approaches. In contrast, each attractor basin is a discrete unit, gates can be used for binary operations (opened/closed, excite/inhibit, update/maintain), and finally "programs" of behavior are stored creating a type of data-code equivalence.

Attractor networks with gating strike a balance between the continuous nature of typical neural networks and the discrete nature of symbolic systems, potentially narrowing the gap between what is possible with systems of each

paradigm. While GALIS attractor networks operate in high-dimensional, continuous space, each attractor within that space can be seen as a discrete “object” (Simen and Polk, 2009). I believe this dual nature of attractor networks presents an underexplored opportunity to produce symbolic-like behaviors using sub-symbolic systems without losing desirable functionality of the sub-symbolic paradigm, such as easy partial pattern matching.

Sequential attractor nets also help to avoid many scaling problems. Because the attractors are sequential rather than fixed points, multiple items can be active “simultaneously” in the same layer (Winder et al., 2009). In fact, the structure of the instruction memory allows multiple sets of multiple items to be activated. This obviates the need to dedicate a network to each possible action by allowing them to be effectively superimposed on a single layer.

Cognitive models built using these GALIS techniques have been able to perform psychological evaluation tasks as well as more familiar tasks, and they do so at the level of human participants despite the high demand these tasks put on executive functioning.

7.2 CONTRIBUTIONS

The work described here makes several contributions to the field.

- The first contribution of this dissertation is the extension of Hopfield-like attractor networks to process sequential rather than fixed-point attractors. This is accomplished with temporally asymmetric weights, which allow the network to act as both be auto- and hetero-associative memories. This new network construct was used to build a model of serial working memory which performed multiple versions of the Running Span task at

levels comparable to human subjects, and exhibited some of the same behavioral patterns such as the recency effect.

- The second contribution presented here is successful combination of multiple networks of this sequential attractor type to create models capable of several cognitive control tasks. This required a method to store not only multiple patterns in the same memory concurrently, but multiple sequences of multiple patterns concurrently, and to do so in a way that allowed the model to store both information about its external environment and information about its task and how to perform it. Such a model was capable, through the manipulation of gated connections between other regions, of using the contents of working memory to autonomously carry out two different cognitive control tasks: the Store/Recognize and the n -Back tasks. Both of these tasks require a network to learn to carry out its own learning of the external environment. In the latter task — n -Back, which is a widely studied benchmark in cognitive psychology — the model was able to switch between task versions without any adjustment to its structure or parameters, even if it was instructed to switch in the middle of task execution. It was also able to match human performance not only in terms of accuracy but also response time.
- The hetero-associativity of asymmetric weights is useful not only for learning sequences, but also for learning links between any elements. This played a major role in the third contribution of this dissertation, in which the GALIS system was enhanced to include visuospatial processing and the binding of different features in a scene. This allows GALIS models to interact with a visuospatial environment rather than passively accepting

a stream of amorphous stimuli. The structure of the visual system is inspired by the dual “what” and “where” pathways of the brain, balances bottom-up and top-down attentional control, and does so via continued use of the gating paradigm. The GALIS model that demonstrated this was tested against results that I collected from human subjects on a card-matching memory test. It was able to match human performance on two versions of the problem, and exceeded the performance of a comparable symbolic model on the more difficult test condition.

- The final contribution was the demonstration that GALIS models with stored instructions are capable of improving their performance as a result of experience with a task. This capability for incremental improvement is one of the major divides between symbolic and sub-symbolic models, and the ability to learn an algorithm like a symbolic system but make marginal improvement through time like a neural network is a significant step forward in narrowing the gap between the two paradigms. This instruction refinement capability was tested on a challenging cognitive psychology task called the Wisconsin Card Sort Test. The WCST requires the binding of multiple visual features, which was possible using the combined symmetric and asymmetric weights of the sequential attractors I developed. It also requires the shifting of attention and inhibition between those features, which was possible by gating inter-region connections, as well as forming a memory of the agent’s own earlier actions, which was possible using the same techniques which my prior models used for memories of stimuli. The model created for the WCST was able to meet all of these requirements the first time it executed the WCST, but was able to do so even

better after it had played several times. This required that the model make incremental improvements to its own internal representations based on nothing more than a binary feedback signal. Further, these improvements came through marginal adjustments to its weights which were originally formed through rapid, one-shot learning procedures. One of the fundamental challenges of the WCST is satisfying the dichotomy created by the stability-plasticity dilemma. The addition of incremental improvements to the mix means that the GALIS model essentially satisfied a trichotomy by striking a balance between not changing, making rapid, punctuational changes, and making slower, more marginal changes.

The work I have contributed to the discipline revolves around the theme of basing neural network behavior on its own memory contents rather than exclusively on network structure. In other words, the theme of my research has been that the storing of programs in neural networks the way they're stored in computers will be effective for implementing cognitive control mechanisms. This is a unique approach in neural networks research, and one that may be viewed as analogous to the shift from special, purpose-built calculating devices to general purpose, von Neumann-type computers. Like that shift, this has the potential to make neural networks more powerful and more re-usable, as well as to provide all of the other benefits that can be derived from shorter development cycle times.

7.3 LIMITATIONS AND FUTURE WORK

From my perspective, there are currently three principal limitations to the GALIS approach. The first is a sensitivity to errors made while recalling

patterns from memory. If a modeler instructs a symbolic system to first look up a location in memory and then output the result he can be confident that, for example, these steps will not happen in the reverse order. If an episodic memory is formed of three events A, B, C occurring in that order, we may be confident a symbolic system will not elide B and skip from A to C. (Note that the same is most certainly *not* true of biological intelligent systems: modern cognitive psychology includes substantial study of biases, errors and weaknesses in human cognition.)

Models created using the GALIS paradigm do not offer the same confidence. There are several potential ways of dealing with this. The work discussed here was built up from the foundation of Hopfield networks. Other, less abstract models of the brain which incorporate spike timing might be more robust to this problem, albeit at the cost of increased computational effort. If more computational energy is to be expended, it is possible that the simple expedient of using larger sequential attractor networks of the same type described here may alleviate this limitation, as capacity increases with network size. The approach used here does recall the correct patterns in the aggregate, so the development of ensembles of sequential attractors that collaborate on recollection of the sequence (perhaps using the same representations and weights but operating from different initial conditions or perhaps each with their own representations and hence weights) should reduce the error rates during recollections. Finally it is worth further testing of these sequential attractor networks with other, similar learning rules such as the Storkey rules (Hu, 2013; Storkey and Valabregue, 1999; Swingler, 2012).

The second limitation of my work is that, although GALIS models can store the algorithm for solving a problem, they cannot develop that algorithm

on their own. The weak link in the chain is the choice of algorithm made by the person designing the system. The ability to hone an instruction set as demonstrated in Chapter 6, while short of being able to generate one from scratch, is a significant move in the right direction and is demonstrative of GALIS' ability to move beyond the explicit instructions given to it by the human modeler. Still needed is the ability to winnow down the useful instructions from a much larger set. This mainly requires a larger initial memory capacity, and the ability to introduce its own instructions or links between them rather than only modifying those that are given.

Related to this would be the addition of a more sophisticated meta-cognitive capability (Cox et al., 2011; Haidarian Shahri et al., 2010; Perlis, 1997). The reinforcement learning-like capability and epi-memory of Chapter 6 opens the door for more sophisticated self-monitoring by the network. This might enable, for example, the model to break out of the unwanted repetitive behaviors it occasionally exhibited with the 16 card trials of Card Matching.

Finally, the structures that can be formed using symmetric and asymmetric weights are somewhat limited. The techniques I have introduced in this dissertation allow multiple sequences of multiple elements each to be stored on the same substrate concurrently. This is even possible with a many-to-many mapping between elements and sequences, i.e., each sequence contains multiple elements, and each element may be a member of more than one sequence. However symbolic AI systems benefit from the ability to use arbitrarily complex data structures. Adapting techniques that can create more complex data structure, such as Holographic Reduced Representations (Harris, 2002; Plate, 2003b), Vector Symbolic Architectures (Levy and Gayler, 2008), or Extended Sparse Distributed Memories (Snaider and Franklin, 2012) would give GALIS

models the ability to implement much more complex algorithms and build more detailed representations of their environments.

Bibliography

- Abbott, A. (2013). Solving the brain. *Nature*, 499:272–274.
- Abbott, L. F. & Blum, K. I. (1991). Functional significance of long-term potentiation for sequence learning and prediction. *Cerebral Cortex*, 6(3):406–416.
- Abbott, L. F. & Song, S. (1999). Temporally asymmetric hebbian learning, spike timing and neuronal response variability. In Kearns, M., Solla, S., & Cohn, D., editors, *Advances in Neural Information Processing Systems 11*, pages 69–75, Cambridge, MA. MIT Press.
- Achlioptas, D. (2003). Database-friendly random projections: Johnson-Lindenstrauss with binary coins. *Journal of Computer and System Sciences*, 66(4):671–687.
- Altmann, E. M. & Gray, W. D. (2002). Forgetting to remember: The functional relationship of decay and interference. *Psychological Science*, 13(1):27–33.
- Amit, D. (1989). *Modeling Brain Function: The World of Attractor Neural Networks*. Cambridge University Press, Cambridge, UK.
- Amos, A. (2000). A computational model of information processing in the frontal cortex and basal ganglia. *Journal of Cognitive Neuroscience*, 12(3):505–19.
- Anderson, J. R. (1996). ACT: A simple theory of complex cognition. *American Psychologist*, 51:355–365.
- Anderson, J. R., Bothell, D., Byrne, M. D., Douglass, S., Lebiere, C., & Qin, Y. (2004). An integrated theory of the mind. *Psychological Review*, 111(4):1036–1060.
- Anderson, J. R. & Lebiere, C. (1998). *The atomic components of thought*. Erlbaum, Mahwah, NJ.
- Anderson, J. R. & Matessa, M. (1997). A production system theory of serial memory. *Psychological Review*, 104(4):728–748.
- Anderson, M. L. (2010). Neural reuse: A fundamental organizational principle of the brain. *Behavioral and Brain Sciences*, 33(4):245–266.
- Awh, E. & Jonides, J. (2001). Overlapping mechanisms of attention and spatial working memory. *Trends in Cognitive Sciences*, 5(3):119–126.

- Baars, B. J. (1983). Conscious contents provide the nervous system with coherent, global information. In Davidson, R. J., Schwartz, G. E., & Shapiro, D., editors, *Consciousness & Self-Regulation*. Plenum Press, New York.
- Baars, B. J. (2002). The conscious access hypothesis: Origins and recent evidence. *Trends in Cognitive Sciences*, 6(1):47–52.
- Baddeley, A. (1968). How does acoustic similarity influence short-term memory? *The Quarterly Journal of Experimental Psychology*, 20(3):249–264.
- Baddeley, A. (2000a). The episodic buffer: A new component of working memory? *Trends in Cognitive Sciences*, 4(11):417–423.
- Baddeley, A. (2000b). Short-term and working memory. In Tulving, E. & Craik, F., editors, *The Oxford Handbook of Memory*, pages 77–92. Oxford University Press.
- Baddeley, A. (2003). Working memory: Looking back and looking forward. *Nature Reviews: Neuroscience*, 4(10):829–839.
- Baddeley, A. (2007). *Working memory, thought, and action*. Number 45 in Oxford Psychology Series. Oxford University Press.
- Baddeley, A. (2012). Working memory: Theories, models, and controversies. *Annual Review of Psychology*, 63:1–29.
- Baddeley, A. & Hitch, G. (1974). Working memory. *The psychology of learning and motivation*, 8:47–89.
- Baizer, J. S., Ungerleider, L. G., & Desimone, R. (1991). Organization of visual inputs to the inferior parietal cortex in macaques. *The Journal of Neuroscience*, 11(1):168–190.
- Bakker, B. (2002). Reinforcement learning with long short-term memory. In Dietterich, T., Becker, S., & Ghahramani, Z., editors, *Advances in Neural Information Processing Systems 14*, pages 1475–1482, Cambridge, MA. MIT Press.
- Barak, O. & Tsodyks, M. (2014). Working models of working memory. *Current Opinion in Neurobiology*, 25:20–24. Theoretical and computational neuroscience.
- Barceló, F. & Knight, R. T. (2002). Both random and perseverative errors underlie WCST deficits in prefrontal patients. *Neuropsychologia*, 40(3):349–356.
- Barnard, P. J. (1985). Interacting cognitive subsystems: A psycholinguistic approach to short-term memory. In Young, A., editor, *Progress in the Psychology of Language*, volume 2, pages 197–258. Erlbaum, London.
- Barnard, P. J. (1999). Interacting Cognitive Subsystems: Modeling working memory phenomena within a multiprocessor architecture. In Miyake, A. & Shah, P., editors, *Models of Working Memory: Mechanisms of Active Maintenance and Executive Control*, chapter 9, pages 298–339. Cambridge University Press.
- Barnard, P. J. & Teasdale, J. D. (1991). Interacting cognitive subsystems: A systemic approach to cognitive-affective interaction and change. *Cognition & Emotion*, 5(1):1–39.

- Barto, A. G. (1995). Adaptive Critics and the Basal Ganglia. In Houk, J., Davis, J., & Beiser, D., editors, *Models of Information Processing in the Basal Ganglia*, pages 215–232. MIT Press, Cambridge, MA.
- Beck, M., Kiani, F., & Bar, B. (2008). Find article from neuron. *Neuron*.
- Beiser, D. G. & Houk, J. C. (1998). Model of cortical-basal ganglionic processing: Encoding the serial order of sensory events. *Journal of Neurophysiology*, 79(6):3168–3188.
- Berg, E. A. (1948). A simple objective technique for measuring flexibility in thinking. *The Journal of General Psychology*, 39(1):15–22.
- Bi, G. & Poo, M. (1998). Synaptic modifications in cultured hippocampal neurons: Dependence on spike timing, synaptic strength, and postsynaptic cell type. *Journal of Neuroscience*, 18(24):10464–10472.
- Bi, G. & Poo, M. (2001). Synaptic modification by correlated activity: Hebb’s Postulate revisited. *Annual Review of Neuroscience*, 24:139–166.
- Blum, K. I. & Abbott, L. F. (1996). A model of spatial map formation in the hippocampus of the rat. *Neural Computation*, 8(1):85–93.
- Boone, K. B. (1999). Neuropsychological assessment of executive functions: Impact of age, education, gender, intellectual level, and vascular status on executive test scores. In Miller, B. L. & Cummings, J. L., editors, *The Science and Practice of Neuropsychology*, pages 247–260. Guilford Press.
- Botvinick, M. & An, J. (2009). Goal-directed decision making in prefrontal cortex: A computational framework. In Koller, D., Schuurmans, D., Bengio, Y., & Bottou, L., editors, *Advances in Neural Information Processing Systems 21*, pages 169–176.
- Botvinick, M. M., Niv, Y., & Barto, A. G. (2009). Hierarchically organized behavior and its neural foundations: A reinforcement learning perspective. *Cognition*, 113(3):262–80.
- Botvinick, M. M. & Plaut, D. C. (2002). Representing task context: Proposals based on a connectionist model of action. *Psychological Research*, 66(4):298–311.
- Botvinick, M. M. & Plaut, D. C. (2006). Short-term memory for serial order: A recurrent neural network model. *Psychological Review*, 113(2):201–233.
- Braver, T. S. & Cohen, J. D. (2000). On the control of control: The role of dopamine in regulating prefrontal function and working memory. In Monsell, S. & Driver, J., editors, *Attention & Performance: Control of Cognitive Processes XVIII*, chapter 31, pages 713–737. MIT Press, Cambridge, MA.
- Bressler, S. & Menon, V. (2010). Large-scale brain networks in cognition. *Trends in Cognitive Sciences*, 14(6):277–290.
- Brown, G. D., Preece, T., & Hulme, C. (2000). Oscillator-based memory for serial order. *Psychological Review*, 107(1):127–181.

- Brown, J. & Braver, T. (2005). Learned predictions of error likelihood in the anterior cingulate cortex. *Science*, 307:1118–1121.
- Bunge, S. (2004). How we use rules to select actions: A review of evidence from cognitive neuroscience. *Cognitive, Affective & Behavioral Neuroscience*, 4:564–579.
- Bunting, M. F., Cowan, N., & Saults, J. S. (2006). How does running memory span work? *Quarterly Journal of Experimental Psychology*, 59(10):1691–700.
- Burgess, N. & Hitch, G. J. (1999). Memory for serial order: A network model of the phonological loop and its timing. *Psychological Review*, 106(3):551–581.
- Burgess, N., Jeffrey, K., & O’Keefe, J. (1999). Integrating hippocampal and parietal functions. In Burgess, N., Jeffrey, K. J., & O’Keefe, J., editors, *The Hippocampal and Parietal Foundations of Spatial Cognition*, pages 2–29. Oxford University Press.
- Chakravarthy, S. & Ghosh, J. (1996). A complex-valued associative memory for storing patterns as oscillatory states. *Biological Cybernetics*, 75:229–238.
- Charniak, E., Riesbeck, C. K., McDermott, D. V., & Meehan, J. R. (2013). *Artificial Intelligence Programming*. Psychology Press, 2nd edition.
- Charron, S. & Koechlin, E. (2010). Divided representation of concurrent goals in the human frontal lobes. *Science*, 328:360–363.
- Chartier, S. & Boukadoum, M. (2006). A bidirectional heteroassociative memory for binary and grey-level patterns. *IEEE Transactions on Neural Networks*, 17(2):385–96.
- Chatham, C. H., Herd, S. A., Brant, A. M., Hazy, T. E., Miyake, A., O’Reilly, R., & Friedman, N. P. (2011). From an executive network to executive control: A computational model of the n -back task. *Journal of Cognitive Neuroscience*, 23(11):3598–619.
- Cho, S. & Reggia, J. A. (1993). Learning competition and cooperation. *Neural Computation*, 5(2):242–259.
- Choo, F.-X. & Eliasmith, C. (2010). A spiking neuron model of serial-order recall. In Cattrambone, R. & Ohlsson, S., editors, *32nd Annual Conference of the Cognitive Science Society*, Portland, OR.
- Cohen, J. D. & O’Reilly, R. C. (1996). A preliminary theory of the interactions between prefrontal cortex and hippocampus that contribute to planning and prospective memory. In Brandimonte, M., Einstein, G. O., & McDaniel, M. A., editors, *Prospective Memory: Theory and Applications*, pages 267–296. Erlbaum, Mahwah, NJ.
- Cohen, M. X. (2008). Neurocomputational mechanisms of reinforcement-guided learning in humans: A review. *Cognitive, Affective, & Behavioral Neuroscience*, 8(2):113–125.

- Compte, A., Brunel, N., Goldman-Rakic, P. S., & Wang, X.-J. (2000). Synaptic mechanisms and network dynamics underlying spatial working memory in a cortical network model. *Cerebral Cortex*, 10(9):910–923.
- Conway, A. R., Cowan, N., Bunting, M. F., Theriault, D. J., & Minkoff, S. R. (2002). A latent variable analysis of working memory capacity, short-term memory capacity, processing speed, and general fluid intelligence. *Intelligence*, 30(2):163–183.
- Courtney, S. M., Ungerleider, L. G., Keil, K., & Haxby, J. V. (1996). Object and spatial visual working memory activate separate neural systems in human cortex. *Cerebral Cortex*, 6(1):39–49.
- Cowan, N. (1988). Evolving conceptions of memory storage, selective attention, and their mutual constraints within the human information-processing system. *Psychological Bulletin*, 104(2):163–191.
- Cowan, N. (1995). *Attention and memory: An integrated framework*. Oxford University Press.
- Cowan, N. (1999). An embedded-processes model of working memory. In Miyake, A. & Shah, P., editors, *Models of working memory: Mechanisms of active maintenance and executive control*, pages 62–101. Cambridge University Press.
- Cowan, N. (2001). The magical number 4 in short-term memory. *Behavioral and Brain Sciences*, 24:87–185.
- Cowan, N., Elliott, E. M., Scott Saults, J., Morey, C. C., Mattox, S., Hismjatullina, A., & Conway, A. R. A. (2005). On the capacity of attention: Its estimation and its role in working memory and cognitive aptitudes. *Cognitive Psychology*, 51(1):42–100.
- Cox, M. T., Oates, T., & Perlis, D. (2011). Toward an integrated metacognitive architecture. In *AAAI Fall Symposium Series: Advances in Cognitive Systems*.
- Dayan, P. (2002). Matters temporal. *Trends in Cognitive Sciences*, 6(3):105–106.
- Dayan, P. (2007). Bilinearity, rules, and prefrontal cortex. *Frontiers in Computational Neuroscience*, 1(1):1–14.
- de Garis, H., Shuo, C., Goertzel, B., & Ruiting, L. (2010). A world survey of artificial brain projects. *Neurocomputing*, 74:3–29.
- Dehaene, S. & Changeux, J.-P. (1989). A Simple Model of Prefrontal Cortex Function in Delayed-Response Tasks. *Journal of Cognitive Neuroscience*, 1(3):244–261.
- Dehaene, S. & Changeux, J.-P. (1991). The Wisconsin Card Sorting Test: Theoretical analysis and modeling in a neuronal network. *Cerebral Cortex*, 1(1):62–79.
- Dehaene, S. & Changeux, J.-P. (1997). A hierarchical neuronal network for planning behavior. *Proceedings of the National Academy of Sciences of the USA*, 94(24):13293–8.
- Dehaene, S. & Changeux, J.-P. (2000). Reward-dependent learning in neuronal networks for planning and decision making. *Progress in Brain Research*, 126:217–

- Dehaene, S., Kerszberg, M., & Changeux, J.-P. (1998). A neuronal model of a global workspace in effortful cognitive tasks. *Proceedings of the National Academy of Sciences of the USA*, 95(24):14529–34.
- Diesmann, M., Gewaltig, M.-O., & Aertsen, A. (1999). Stable propagation of synchronous spiking in cortical neural networks. *Nature*, 402:529–533.
- Dougherty, M. R. & Hunter, J. E. (2003a). Hypothesis generation, probability judgment, and individual differences in working memory capacity. *Acta Psychologica*, 113(3):263–282.
- Dougherty, M. R. & Hunter, J. E. (2003b). Probability judgment and subadditivity: The role of working memory capacity and constraining retrieval. *Memory & Cognition*, 31(6):968–982.
- Doya, K. (1999). What are the computations of the cerebellum, the basal ganglia and the cerebral cortex? *Neural Networks*, 12(7–8):961–974.
- Durstewitz, D. & Seamans, J. K. (2008). The dual-state theory of prefrontal cortex dopamine function with relevance to catechol-o-methyltransferase genotypes and schizophrenia. *Biological Psychiatry*, 64(9):739–49.
- Durstewitz, D., Seamans, J. K., & Sejnowski, T. J. (2000). Neurocomputational models of working memory. *Nature Neuroscience*, 3(11supp.):1184–1191.
- Eliasmith, C. (2005). Cognition with neurons: A large-scale, biologically realistic model of the Wason task. In Bara, G., Barsalou, L., & Bucciarelli, M., editors, *Proceedings of the 27th Annual Meeting of the Cognitive Science Society*, pages 624–629, Stresa, Italy.
- Eliasmith, C. (2013). *How to build a brain: A neural architecture for biological cognition*. Oxford University Press.
- Eliasmith, C. & Anderson, C. H. (2003). *Neural Engineering: Computation, representation and dynamics in neurobiological systems*. MIT Press.
- Eliasmith, C., Stewart, T. C., Choo, X., Bekolay, T., DeWolf, T., Tang, Y., & Rasmussen, D. (2012). A large-scale model of the functioning brain. *Science*, 338(6111):1202–1205.
- Elman, J. L. (1990). Finding structure in time. *Cognitive Science*, 14(2):179–211.
- Engle, R. W., Conway, A. R. A., Tuholski, S. W., & Shisler, R. J. (1995). A resource account of inhibition. *Psychological Science*, 6(2):122–125.
- Engle, R. W., Tuholski, S. W., Laughlin, J. E., & Conway, A. R. A. (1999). Working memory, short-term memory, and general fluid intelligence: A latent-variable approach. *Journal of Experimental Psychology*, 128(3):309–331.
- Everett, J., Lavoie, K., Gagnon, J.-F., & Gosselin, N. (2001). Performance of patients with schizophrenia on the Wisconsin Card Sorting Test. *Journal of Psychiatry and Neuroscience*, 26(2):123.

- Farrell, S. & Lewandowsky, S. (2002). An endogenous distributed model of ordering in serial recall. *Psychonomic Bulletin & Review*, 9(1):59–79.
- Feldman, J. (2013). The neural binding problem. *Cognitive Neurodynamics*, 7(1):1–11.
- Forgy, C. L. (1982). Rete: A fast algorithm for the many pattern/many object pattern match problem. *Artificial Intelligence*, 19(1):17–37.
- Frank, M. J. (2005). Dynamic dopamine modulation in the basal ganglia: A neurocomputational account of cognitive deficits in medicated and nonmedicated Parkinsonism. *Journal of cognitive neuroscience*, 17(1):51–72.
- Frank, M. J., Loughry, B., & O’Reilly, R. C. (2001). Interactions between frontal cortex and basal ganglia in working memory: A computational model. *Cognitive, Affective & Behavioral Neuroscience*, 1:137–160.
- Gayler, R. W. (2003). Vector Symbolic Architectures answer Jackendoff’s challenges for cognitive neuroscience. In Slezak, P., editor, *ICCS/ASCS International Conference on Cognitive Science*, pages 133–138, Sydney, Australia.
- Gazzaley, A. & Nobre, A. C. (2012). Top-down modulation: Bridging selective attention and working memory. *Trends in Cognitive Sciences*, 16(2):129 – 135.
- Gers, F. A. & Schmidhuber, J. (2000). Recurrent nets that time and count. In *Proceedings of the International Joint Conference on Neural Networks*, pages 189–194.
- Gers, F. A., Schmidhuber, J., & Cummins, F. (2000). Learning to forget: Continual prediction with LSTM. *Neural Computation*, 12(10):2451–71.
- Gerstner, W. & Kistler, W. M. (2002). Mathematical formulations of Hebbian learning. *Biological Cybernetics*, 87(5–6):404–415.
- Gerstner, W., Ritz, R., & van Hemmen, J. L. (1993). Why spikes? Hebbian learning and retrieval of time-resolved excitation patterns. *Biological Cybernetics*, 69(5–6):503–515.
- Goel, V., Pullara, S. D., & Grafman, J. (2001). A computational model of frontal lobe dysfunction: working memory and the Tower of Hanoi task. *Cognitive Science*, 25(2):287–313.
- Goldman-Rakic, P. S. (1987). Circuitry of primate prefrontal cortex and regulation of behavior by representational memory. *Handbook of Physiology — The Nervous System*, 5:373–417.
- Gomez, F. & Schmidhuber, J. (2005). Evolving modular fast-weight networks for control. In *Proceedings 15th International Conference Artificial Neural Networks, Part II: Formal Models and Their Applications (LNCS 3697)*, pages 383–389. Springer.
- Greve, K. W., Stickle, T. R., Love, J. M., Bianchini, K. J., & Stanford, M. S. (2005). Latent structure of the Wisconsin Card Sorting Test: A confirmatory factor analytic study. *Archives of Clinical Neuropsychology*, 20(3):355–364.

- Guigon, E., Dorizzi, B., Burnod, Y., & Schultz, W. (1995). Neural correlates of learning in the prefrontal cortex of the monkey: A predictive model. *Cerebral Cortex*, 5(2):135–147.
- Gurney, K., Prescott, T. J., & Redgrave, P. (2001a). A computational model of action selection in the basal ganglia, I: A new functional anatomy. *Biological Cybernetics*, 84(6):401–410.
- Gurney, K., Prescott, T. J., & Redgrave, P. (2001b). A computational model of action selection in the basal ganglia, II: Analysis and simulation of behaviour. *Biological Cybernetics*, 84(6):411–423.
- Haarman, H. & Usher, M. (2001). Maintenance of semantic information in capacity-limited short-term memory. *Psychonomic Bulletin*, 8(3):568–578.
- Haidarian Shahri, H., Dinalankara, W., Fults, S., Wilson, S., Perlis, D., Schmill, M., Oates, T., Josyula, D., & Anderson, M. (2010). The metacognitive loop: An architecture for building robust intelligent systems. In *AAAI Fall Symposium Series: Commonsense Knowledge*.
- Harbison, J. I., Atkins, S. M., & Dougherty, M. R. (2011). Performance gains in an adaptive n -back working memory training task. In *Proceedings 50th Annual Meeting Psychonomic Society*, pages 120–125.
- Harris, H. (2002). Holographic reduced representations for oscillator recall: A model of phonological production. In Gray, W. D. & Schunn, C. D., editors, *Proceedings of the 24th Annual Conference of the Cognitive Science Society*. Erlbaum.
- Hasselmo, M., Bodelon, C., & Wyble, B. (2002). Proposed function for hippocampal theta rhythm. *Neural Computation*, 14:793–817.
- Hayashi, Y. (1994). Oscillatory neural network and learning of continuously transformed patterns. *Neural Networks*, 7:219–231.
- Hazy, T. E., Frank, M. J., & O’Reilly, R. C. (2007). Towards an executive without a homunculus: Computational models of the prefrontal cortex/basal ganglia system. *Philosophical Transactions of the Royal Society B*, 362(1485):1601–13.
- Henson, R. N. (1998). Short-term memory for serial order: The start-end model. *Cognitive Psychology*, 36(2):73–137.
- Henson, R. N. & Burgess, N. (1997). Representations of serial order. In *4th Neural Computation and Psychology Workshop: Connectionist Representations*, pages 283–300. Springer Verlag.
- Henson, R. N., Norris, D. G., Page, M. P., & Baddeley, A. (1996). Unchained memory: Error patterns rule out chaining models of immediate serial recall. *The Quarterly Journal of Experimental Psychology*, 49A(1):80–115.
- Hinton, G. E. (2002). Training products of experts by minimizing contrastive divergence. *Neural Computation*, 14(8):1771–1800.

- Hinton, G. E. & Plaut, D. C. (1987). Using fast weights to deblur old memories. In *Program of the Ninth Annual Conference of the Cognitive Science Society*, pages 177–186.
- Hochreiter, S. & Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, 9(8):1735–80.
- Hockey, A. & Geffen, G. (2004). The concurrent validity and test–retest reliability of a visuospatial working memory task. *Intelligence*, 32(6):591–605.
- Holyoak, K. J. & Hummel, J. E. (2000). The proper treatment of symbols in a connectionist architecture. *Cognitive dynamics: Conceptual change in humans and machines*, pages 229–263. NOT CITED YET.
- Hopfield, J. J. (1982). Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences of the USA*, 79(8):2554–2558.
- Hopfield, J. J., Feinstein, D. I., & Palmer, R. G. (1983). ‘Unlearning’ has a stabilizing effect in collective memories. *Nature*, 304(5922):158–159.
- Horn, D. & Usher, M. (1991). Parallel activation of memories in an oscillatory neural network. *Neural Computation*, 3:31–43.
- Horn, D. & Usher, M. (1992). Oscillatory model of short term memory. In Moody, J. E., Hanson, S. J., & Lippmann, R. P., editors, *Advances in Neural Information and Processing Systems 4*, pages 125–132. Morgan Kaufmann.
- Hoshino, O., Usuba, N., Kashimori, Y., & Kambara, T. (1997). Role of itinerancy among attractors as dynamical map in distributed coding scheme. *Neural Networks*, 10(8):1375–1390.
- Houghton, G. (1990). The problem of serial order: A neural network model of sequence learning and recall. In Dale, R., Mellish, C., & Zock, M., editors, *Current Research in Natural Language Generation*, pages 287–319. London Academic Press.
- Houk, J. C. (2007). Biological implementation of the temporal difference algorithm for reinforcement learning: theoretical comment on O’Reilly et al. (2007). *Behavioral neuroscience*, 121(1):231–2.
- Hsiang, S. M. (2013). Visually-weighted regression. *SSRN 2265501*.
- Hu, X. (2013). Storkey learning rules for Hopfield networks. *viXra*, 1309.0130.
- Ingber, L. (1995). Statistical mechanics of neocortical interactions: Constraints on 40 Hz models of short term memory based on persistent spiking. *Physical Review E*, 52:4561–4563.
- Jacobs, R. A., Jordan, M. I., & Barto, A. G. (1991a). Task decomposition through competition in a modular connectionist architecture: The what and where vision tasks. *Cognitive Science*, 15(2):219–250.
- Jacobs, R. A., Jordan, M. I., Nowlan, S. J., & Hinton, G. E. (1991b). Adaptive mixtures of local experts. *Neural Computation*, 3(1):79–87.

- Jaeger, H. (2001). The “echo state” approach to analysing and training recurrent neural networks. Technical Report 148, German National Research Center for Information Technology.
- Jaeggi, S. M., Buschkuhl, M., Jonides, J., & Perrig, W. J. (2008). Improving fluid intelligence with training on working memory. *Proceedings of the National Academy of Sciences of the USA*, 105(19):6829–6833.
- Jaeggi, S. M., Buschkuhl, M., Jonides, J., & Shah, P. (2011). Short- and long-term benefits of cognitive training. *Proceedings of the National Academy of Sciences*, 108(25):10081–10086.
- Jaeggi, S. M., Studer-Luethi, B., Buschkuhl, M., Su, Y.-F., Jonides, J., & Perrig, W. J. (2010). The relationship between n -back performance and matrix reasoning: Implications for training and transfer. *Intelligence*, 38(6):625–635.
- Jilk, D. J., Lebiere, C., O’Reilly, R. C., & Anderson, J. R. (2008). SAL: An explicitly pluralistic cognitive architecture. *Journal of Experimental & Theoretical Artificial Intelligence*, 20(3):197–218.
- Johnson, W. B. & Lindenstrauss, J. (1984). Extensions of Lipschitz mappings into a Hilbert space. *Contemporary Mathematics*, 26(189–206):1.
- Jones, M. & Polk, T. A. (2002). An attractor network model of serial recall. *Cognitive Systems Research*, 3(1):45–55.
- Jordan, M. I. & Jacobs, R. A. (1994). Hierarchical mixtures of experts and the EM algorithm. *Neural Computation*, 6(2):181–214.
- Kak, S. (1993). On training feedforward neural networks. *Pramana*, 40(1):35–42.
- Kak, S. (1998). On generalization by neural networks. *Information Sciences*, 111(1–4):293–302.
- Kak, S. (1999). Better web searches and prediction with instantaneously trained neural networks. *IEEE Intelligent Systems*, 14(6):78–81.
- Kak, S. (2002). A class of instantaneously trained neural networks. *Information Sciences*, 148(1–4):97–102.
- Kaplan, G., Sengör, N. S., Gürvit, H., Genç, I., & Güzeli, C. (2006). A composite neural network model for perseveration and distractibility in the Wisconsin Card Sorting Test. *Neural Networks*, 19(4):375–387.
- Katori, Y., Sakamoto, K., Saito, N., Tanji, J., Mushiake, H., & Aihara, K. (2011). Representational switching by dynamical reorganization of attractor structure in a network model of the prefrontal cortex. *PLoS Computational Biology*, 7(11):e1002266.
- Kesner, R. P., Gilbert, P. E., & Wallenstein, G. V. (2000). Testing neural network models of memory with behavioral experiments. *Current Opinion in Neurobiology*, 10(2):260–5.
- Kiela, D. (2011). *Variable Binding in Biologically Plausible Neural Networks*. PhD thesis, Universiteit van Amsterdam.

- Koene, R. & Hasselmo, M. (2007). First-in-first-out item replacement in a model of short-term memory based on persistent spiking. *Cerebral Cortex*, 17:1766–1781.
- Kohonen, T. (1998). The self-organizing map. *Neurocomputing*, 21(1–3):1–6.
- Korsten, N. J. H., Fragopanagos, N., Hartley, M., Taylor, N., & Taylor, J. G. (2006). Attention as a controller. *Neural Networks*, 19(9):1408–1421.
- Kremer, S. C. (2001). Spatiotemporal connectionist networks: A taxonomy and review. *Neural Computation*, 13(2):249–306.
- Kriete, T., Noelle, D. C., Cohen, J. D., & O'Reilly, R. C. (2013). Indirection and symbol-like processing in the prefrontal cortex and basal ganglia. *Proceedings of the National Academy of Sciences of the USA*, 110(41):16390–16395.
- Kulesza, A. & Taskar, B. (2010). Structured determinantal point processes. In *Advances in Neural Information Processing Systems 22*, pages 1171–1179.
- Kulesza, A. & Taskar, B. (2012). Determinantal point processes for machine learning. *arXiv preprint 1207.6083*.
- Lamb, L. d. C. (2008). The grand challenges and myths of neural-symbolic computation. In Raedt, L. D., Hammer, B., Hitzler, P., & Maass, W., editors, *Recurrent Neural Networks: Models, Capacities, and Applications*, number 08041 in Dagstuhl Seminar Proceedings, Dagstuhl, Germany.
- Langner, R., Sternkopf, M. A., Kellermann, T. S., Grefkes, C., Kurth, F., Schneider, F., Zilles, K., & Eickhoff, S. B. (2013). Translating working memory into action: Behavioral and neural evidence for using motor representations in encoding visuo-spatial sequences. *Human Brain Mapping*, 35(7):3465–3484.
- Le, H. S. (2012). *Continuous space models with neural networks in natural language processing*. PhD thesis, Université Paris Sud.
- Lebiere, C. & Anderson, J. R. (1993). A connectionist implementation of the ACT-R production system. In *Proceedings of the 15th Annual Conference of the Cognitive Science Society*, pages 635–640.
- Levy, S. D. & Gayler, R. (2008). Vector symbolic architectures: A new building material for artificial general intelligence. In *Proceedings of the Conference on Artificial General Intelligence*, pages 414–418. IOS Press.
- Lewandowsky, S. (1999). Redintegration and response suppression in serial recall: A dynamic network model. *International Journal of Psychology*, 34(5):434–446.
- Lewandowsky, S. & Farrell, S. (2003). Computational models of working memory. In Nadel, L., editor, *Encyclopedia of Cognitive Science*. Macmillan Reference.
- Lewandowsky, S. & Farrell, S. (2008). Short-term memory: New data and a model. *Psychology of Learning and Motivation*, 49:1–48.
- Lewandowsky, S. & Murdock, B. B. (1989). Memory for serial order. *Psychological Review*, 96(1):25–57.

- Li, S.-C. & Lewandowsky, S. (1993). Intralist distractors and recall: Constraints on models of memory for serial order. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, 19(4):895–908.
- Lipson, H. & Siegelmann, H. (2000). Clustering irregular shapes using high-order neurons. *Neural Computation*, 12(10):2331–2353.
- Lisman, J. & Idiart, M. (1995). Storage of 7 ± 2 short-term memories in oscillatory subcycles. *Science*, 267:1512–6.
- Long, D. L., Parks, R. W., & Levine, D. S. (1998). An introduction to neural network modeling: Merits, limitations, and controversies. In Parks, R. W., Levine, D. S., & Long, D. L., editors, *Fundamentals of Neural Network Modeling*, pages 3–32. MIT Press, Cambridge, MA.
- Ma, J. (1999). The asymptotic memory capacity of the generalized Hopfield network. *Neural Networks*, 12(9):1207–1212.
- MacDonald, A., Cohen, J., Stenger, V., & Carter, C. (2000). Dissociating the role of the dorsolateral prefrontal and anterior cingulate cortex in cognitive control. *Science*, 288:1835–1838.
- Maniadakis, M., Trahanias, P., & Tani, J. (2012). Self-organizing high-order cognitive functions in artificial agents: Implications for possible prefrontal cortex mechanisms. *Neural Networks*, 33:76–87.
- Marcus, G. F. (2001). *The Algebraic Mind: Integrating Connectionism and Cognitive Science*. MIT Press, Cambridge, MA.
- Markram, H., Lubke, J., Frotscher, M., & Sakmann, B. (1997). Regulation of synaptic efficacy by coincidence of postsynaptic APs and EPSPs. *Science*, 275:213–215.
- Martinet, L.-E., Sheynikhovich, D., Benchenane, K., & Arleo, A. (2011). Spatial learning and action planning in a prefrontal cortical network model. *PLoS Computational Biology*, 7(5).
- McCarthy, G., Puce, A., Constable, R. T., Krystal, J. H., Gore, J. C., & Goldman-Rakic, P. S. (1996). Activation of human prefrontal cortex during spatial and nonspatial working memory tasks measured by functional MRI. *Cerebral Cortex*, 6(4):600–11.
- McEliece, R., Posner, E., Rodemich, E., & Venkatesh, S. (1987). The capacity of the Hopfield associative memory. *IEEE Transactions Information Theory*, 33(4):461–482.
- Miller, G. A., Galanter, E., & Pribram, K. H. (1960). *Plans and the Structure of Behavior*. Holt, Rinehart and Winston, New York.
- Milner, B. (1963). Effects of different brain lesions on card sorting: The role of the frontal lobes. *Archives of Neurology*, 9(1):90–100.
- Miranker, D. P. (1987). TREAT: A better match algorithm for ai production systems. Technical Report AI TR87-58, University of Texas at Austin.

- Monchi, O. (2000). A neural model of working memory processes in normal subjects, Parkinson’s disease and schizophrenia for fMRI design and predictions. *Neural Networks*, 13(8–9):953–973.
- Monchi, O., Petrides, M., Petre, V., Worsley, K., & Dagher, A. (2001). Wisconsin Card Sorting revisited: Distinct neural circuits participating in different stages of the task identified by event-related functional magnetic resonance imaging. *The Journal of Neuroscience*, 21(19):7733–41.
- Monchi, O. & Taylor, J. G. (1999). A hard wired model of coupled frontal working memories for various tasks. *Information Sciences*, 113(3–4):221–243.
- Monner, D. & Reggia, J. (2013). Emergent latent symbol systems in recurrent neural networks. *Connection Science*, 12:1932–1943.
- Monner, D. & Reggia, J. A. (2012). A generalized LSTM-like training algorithm for second-order recurrent neural networks. *Neural Networks*, 25:70–83.
- Morris, G., Nevet, A., Arkadir, D., Vaadia, E., & Bergman, H. (2006). Mid-brain dopamine neurons encode decisions for future action. *Nature Neuroscience*, 9(8):1057–1063.
- Morton, J. B. & Munakata, Y. (2001). Active versus latent representations: A neural network model of perseveration, dissociation, and decalage. *Developmental Psychobiology*, 40(3):255–265.
- Navarro, G. (2001). A guided tour to approximate string matching. *ACM Computing Surveys*, 33(1):31–88.
- Nelson, H. E. (1976). A modified card sorting test sensitive to frontal lobe defects. *Cortex*, 12(4):313–324.
- Newell, A. & Simon, H. A. (1972). *Human Problem Solving*. Prentice-Hall.
- O’Doherty, J., Dayan, P., Schultz, J., Deichmann, R., Friston, K. J., & Dolan, R. J. (2004). Dissociable roles of ventral and dorsal striatum in instrumental conditioning. *Science*, 304(5669):452–454.
- Omlin, C. W. & Giles, L. (2000). Symbolic knowledge representation in recurrent neural networks: Insights from theoretical models of computation. In Cloete, I. & Zuruda, J. M., editors, *Knowledge Based Neurocomputing*, chapter 3, pages 63–115. MIT Press, Cambridge, MA.
- O’Reilly, R., Hazy, T., & Mollick, J. (2014). Goal-driven cognition in the brain: A computational framework. *arXiv preprint arXiv:*
- O’Reilly, R. C. (2006). Biologically based computational models of high-level cognition. *Science*, 314(5796):91.
- O’Reilly, R. C., Braver, T. S., & Cohen, J. D. (1997). A biologically-based computational model of working memory. In Miyake, A. & Shah, P., editors, *Models of Working Memory: Mechanisms of Active Maintenance and Executive Control*. Cambridge University Press, New York.

- O'Reilly, R. C. & Busby, R. S. (2002). Generalizable relational binding from coarse-coded distributed representations. In Dietterich, T., Becker, S., & Ghahramani, Z., editors, *Advances in Neural Information Processing Systems 14*, pages 75–82.
- O'Reilly, R. C. & Frank, M. J. (2006). Making working memory work: A computational model of learning in the prefrontal cortex and basal ganglia. *Neural Computation*, 18(2):283–328.
- O'Reilly, R. C., Frank, M. J., Hazy, T. E., & Watz, B. (2007). PVLV: The primary value and learned value Pavlovian learning algorithm. *Behavioral Neuroscience*, 121(1):31–49.
- O'Reilly, R. C., Herd, S. A., & Pauli, W. M. (2010). Computational models of cognitive control. *Current Opinion in Neurobiology*, 20(2):257–61.
- O'Reilly, R. C., Noelle, D. C., Braver, T. S., & Cohen, J. D. (2002). Prefrontal cortex and dynamic categorization tasks: Representational organization and neuromodulatory control. *Cerebral Cortex*, 12(3):246–57.
- Owen, A. M., McMillan, K. M., Laird, A. R., & Bullmore, E. (2005). *N*-back working memory paradigm: A meta-analysis of normative functional neuroimaging studies. *Human Brain Mapping*, 25(1):46–59.
- Page, M. P. & Norris, D. (1998). The primacy model: A new model of immediate serial recall. *Psychological Review*, 105(4):761–781.
- Pan, F., Roberts, A., McMillan, L., de Villena, F. P. M., Threadgill, D., & Wang, W. (2007). Sample selection for maximal diversity. In *Seventh IEEE International Conference on Data Mining (ICDM)*, pages 262–271.
- Pantic, L., Torres, J. J., Kappen, H. J., & Gielen, S. C. (2002). Associative memory with dynamic synapses. *Neural Computation*, 14(12):2903–2923.
- Pascanu, R. & Jaeger, H. (2011). A neurodynamical model for working memory. *Neural Networks*, 24(2):199–207.
- Perlis, D. (1997). Consciousness as self-function. *Journal of Consciousness Studies*, 4(5-6):509–525.
- Petrides, M. (1994). Frontal lobes and behaviour. *Current Opinion in Neurobiology*, 4(2):207–211.
- Plate, T. A. (1995). Holographic reduced representations. *IEEE Transactions on Neural Networks*, 6(3):623–41.
- Plate, T. A. (2003a). Distributed representations. In Nadel, L., editor, *Encyclopedia of Cognitive Science*. Macmillan Reference, London.
- Plate, T. A. (2003b). *Holographic Reduced Representation: Distributed Representation for Cognitive Structures*. Center for the Study of Language and Information, Stanford, CA.
- Polk, T. A., Simen, P., Lewis, R. L., & Freedman, E. (2002). A computational approach to control in complex cognition. *Cognitive Brain Research*, 15(1):71–83.

- Pollack, I., Johnson, I., & Knaff, P. (1959). Running memory span. *Journal of Experimental Psychology*, 57:137–146.
- Ponzi, A. (2008). Dynamical model of salience gated working memory, action selection and reinforcement based on basal ganglia and dopamine feedback. *Neural Networks*, 21(2–3):322–330.
- Raffone, A. & Wolters, G. (2001). A cortical mechanism for binding in visual working memory. *Journal of Cognitive Neuroscience*, 13:766–785.
- Rahimi, A. & Recht, B. (2007). Random features for large-scale kernel machines. In *Advances Neural Information Processing Systems 20*, pages 1177–1184.
- Rahimi, A. & Recht, B. (2008). Weighted sums of random kitchen sinks: Replacing minimization with randomization in learning. In *Advances in Neural Information Processing Systems 21*, pages 1313–1320.
- Rao, R. P. & Sejnowski, T. J. (2001). Spike-timing-dependent Hebbian plasticity as temporal difference learning. *Neural Computation*, 13(10):2221–2237.
- Rasmussen, D. & Eliasmith, C. (2010). A neural model of rule generation in inductive reasoning. In Ohlsson, R. & Cattrambone, S., editors, *Proceedings of the 32nd Annual Conference of the Cognitive Science Society*.
- Redgrave, P., Prescott, T. J., & Gurney, K. N. (1999). The basal ganglia: A vertebrate solution to the selection problem? *Neuroscience*, 89(4):1009–23.
- Reggia, J. A., D’Autrechy, C. L., Sutton, G. G., & Weinrich, M. (1992). A competitive distribution theory of neocortical dynamics. *Neural Computation*, 4(3):287–317.
- Reggia, J. A. & Edwards, M. (1990). Phase transitions in connectionist models having rapidly varying connection strengths. *Neural Computation*, 2(4):523–535.
- Reggia, J. A., Marsland, P. M., & Sloan, R. B. (1988). Competitive dynamics in a dual-route connectionist model of print-to-sound transformation. *Complex Systems*, 2(5):509–547.
- Reggia, J. A., Monner, D., & Sylvester, J. C. (2014). The computational explanatory gap. *Journal of Consciousness Studies*, 21(9–10): *in press*.
- Reggia, J. A., Sylvester, J. C., Weems, S. A., & Bunting, M. F. (2009). A simple oscillatory short-term memory. In *Biologically Inspired Cognitive Architectures II*, pages 103–108.
- Reynolds, J. H. & Desimone, R. (1999). The role of neural mechanisms of attention in solving the binding problem. *Neuron*, 24(1):19–29, 111–25.
- Riachi, I., Hill, S., Schrmann, F., & Markram, H. (2009). Capturing neuron morphological diversity. In De Schutter, E., editor, *Computational Modeling Methods for Neuroscientists*. MIT Press.
- Richardson, J. T. E., Engle, R. W., Hasher, L., Logie, R. H., Stoltzfus, E. R., & Zacks, R. T. (1996). *Working Memory and Human Cognition*. Oxford University Press.

- Rivest, F., Bengio, Y., & Kalaska, J. (2004). Brain inspired reinforcement learning. In Saul, L. K., Weiss, Y., & Bottou, L., editors, *Advances in Neural Information Processing Systems 17*, pages 1129–1136.
- Roesch, M. R., Calu, D. J., & Schoenbaum, G. (2007). Dopamine neurons encode the better option in rats deciding between differently delayed or sized rewards. *Nature Neuroscience*, 10(12):1615–1624.
- Romine, C. B., Lee, D., Wolfe, M. E., Homack, S., George, C., & Riccio, C. A. (2004). Wisconsin Card Sorting Test with children: A meta-analytic study of sensitivity and specificity. *Archives of Clinical Neuropsychology*, 19(8):1027–1041.
- Ronco, E., Gawthrop, P., & Hill, D. (1998). Gated modular neural networks for control oriented modelling. Technical Report EE-98009, University of Sydney.
- Ronco, E. & Gawthrop, P. J. (1997). Neural networks for modelling and control. Technical Report csc97008, University of Glasgow.
- Rougier, N. P., Noelle, D. C., Braver, T. S., Cohen, J. D., & O’Reilly, R. C. (2005). Prefrontal cortex and flexible cognitive control: Rules without symbols. *Proceedings of the National Academy of Sciences of the USA*, 102(20):7338–43.
- Roy, A. (2008). Connectionism, controllers, and a brain theory. *IEEE Transactions on Systems, Man and Cybernetics, Part A*, 38:1434–1441.
- Ruiz, M. & Srinivasan, P. (2002). Hierarchical text categorization using neural networks. *Information Retrieval*, 5(1):87–118.
- Samadi, M., Felner, A., & Schaeffer, J. (2008). Learning from multiple heuristics. In *Proceedings of the 23rd National Conference on Artificial intelligence (AAAI)*, pages 357–362.
- Sandberg, A., Tegnér, J., & Lansner, A. (2003). A working memory model based on fast Hebbian learning. *Network: Computation in Neural Systems*, 14(4):789–802.
- Schmajuk, N. A. & Thieme, A. D. (1992). Purposive behavior and cognitive mapping: a neural network model. *Biological Cybernetics*, 67:165–174.
- Schmidhuber, J. (1992). Learning to control fast-weight memories: An alternative to dynamic recurrent networks. *Neural Computation*, 4(1):131–139.
- Schmidhuber, J. (2014). Deep learning in neural networks: An overview. *arXiv preprint 1404.7828*.
- Schmidt, H., Jogia, J., Fast, K., Christodoulou, T., Haldane, M., Kumari, V., & Frangou, S. (2009). No gender differences in brain activation during the n -back task: An fMRI study in healthy individuals. *Human Brain Mapping*, 30(11):3609–3615.
- Schneider, W. (1999). Working memory in a multilevel hybrid connectionist control architecture (CAP2). In Miyake, A. & Shah, P., editors, *Models of Working Memory: Mechanisms of Active Maintenance and Executive Control*, chapter 10, pages 340–374. Cambridge University Press.

- Schneider, W. & Chein, J. M. (2003). Controlled & automatic processing: Behavior, theory, and biological mechanisms. *Cognitive Science*, 27(3):525–559.
- Schneider, W. & Detweiler, M. (1987). A connectionist/control architecture for working memory. *The Psychology of Learning and Motivation*, 21:53–119.
- Schneider, W. & Oliver, W. L. (1989). An intractable connectionist/control architecture: Using rule-based instructions to accomplish connectionist learning in a human time scale. Technical Report AIP-95, Carnegie Mellon University, Pittsburgh, PA.
- Schoofs, D., Preuß, D., & Wolf, O. T. (2008). Psychosocial stress induces working memory impairments in an *n*-back paradigm. *Psychoneuroendocrinology*, 33(5):643–53.
- Schroll, H., Vitay, J., & Hamker, F. H. (2012). Working memory and response selection: A computational account of interactions among cortico-basalganglio-thalamic loops. *Neural Networks*, 26:59–74.
- Schulz, R. & Reggia, J. A. (2004). Temporally asymmetric learning supports sequence processing in multi-winner self-organizing maps. *Neural Computation*, 16:535–561.
- Shedden, J. & Schneider, W. (1990). A connectionist model of attentional enhancement and signal buffering. Technical Report AIP-123, University of Pittsburgh, Pittsburgh, PA.
- Sherman, S. & Guillery, R. (2009). *Exploring the Thalamus and Its Role in Cortical Function*. MIT Press, 2nd edition.
- Simen, P. & Polk, T. A. (2009). A symbolic/subsymbolic interface protocol for cognitive modeling. *Logic Journal of the IGPL*, 18(5):705–761.
- Simen, P., Vugt, M. V., Balci, F., Freestone, D., & Polk, T. A. (2010). Toward an analog neural substrate for production systems. In *Proceedings of the 10th International Conference on Cognitive Modeling*, pages 223–228.
- Singer, W. (2011). Dynamic formation of functional networks by synchronization. *Neuron*, 69:191–193.
- Smith, E. E., Jonides, J., Koeppe, R. A., Awh, E., Schumacher, E. H., & Minoshima, S. (1995). Spatial versus Object Working Memory: PET Investigations. *Journal of Cognitive Neuroscience*, 7(3):337–356.
- Smith-Seemiller, L., Arffa, S., & Franzen, M. D. (2001). Use of Wisconsin Card Sorting Test short forms with school-age children. *Archives of Clinical Neuropsychology*, 16(5):489–499.
- Smolensky, P., Goldrick, M., & Mathis, D. (2014). Optimization and quantization in gradient symbol systems: A framework for integrating the continuous and the discrete in cognition. *Cognitive science*, 38(6):1102–1138.
- Snaider, J. & Franklin, S. (2012). Extended sparse distributed memory and sequence storage. *Cognitive Computation*, 4(2):172–180.

- Socher, R., Huval, B., Manning, C. D., & Ng, A. Y. (2012). Semantic compositionality through recursive matrix-vector spaces. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 1201–1211. Association for Computational Linguistics.
- Sporns, O. (2011). *Networks of the Brain*. MIT Press.
- Sprenger, A. M., Atkins, S. M., Bolger, D. J., Harbison, J., Novick, J. M., Chrabaszcz, J. S., Weems, S. A., Smith, V., Bobb, S., Bunting, M. F., & Dougherty, M. R. (2013). Training working memory: Limits of transfer. *Intelligence*, 41(5):638–663.
- Stemme, A., Deco, G., & Busch, A. (2007). The neuronal dynamics underlying cognitive flexibility in set shifting tasks. *Journal of Computational Neuroscience*, 23(3):313–331.
- Stewart, T. C., Choo, F.-X., & Eliasmith, C. (2010). Symbolic reasoning in spiking neurons: A model of the cortex/basal ganglia/thalamus loop. In *Proceedings of the 32nd Annual Conference of the Cognitive Science Society*, pages 1100–1105.
- Stewart, T. C. & Eliasmith, C. (2011). Neural cognitive modelling: A biologically constrained spiking neuron model of the tower of hanoi task. In *Proceedings 33rd Annual Conference Cognitive Science Society*, pages 565–661.
- Stewart, T. C., Tang, Y., & Eliasmith, C. (2011). A biologically realistic cleanup memory: Autoassociation in spiking neurons. *Cognitive Systems Research*, 12(2):84–92.
- Storkey, A. (1997). Increasing the capacity of a Hopfield network without sacrificing functionality. *Proceedings International Conference Artificial Neural Networks (ICANN)*, pages 451–456.
- Storkey, A. & Valabregue, R. (1999). The basins of attraction of a new Hopfield learning rule. *Neural Networks*, 12(6):869–876.
- Strauss, E., Sherman, E. M. S., & Otfried, S. (2006). *A compendium of neuropsychological tests: Administration, norms, and commentary*. Oxford University Press, 3rd edition.
- Stuart, G. & Hulme, C. (2009). Lexical and semantic influences on immediate serial recall: A role for redintegration. In Thorn, A. & Page, M., editors, *Interactions between short-term and long-term memory in the verbal domain*, pages 157–176. Psychology Press.
- Sun, R. (2006). The CLARION cognitive architecture: Extending cognitive modeling to social simulation. In Sun, R., editor, *Cognition and Multi-Agent Interaction*, pages 79–99. Cambridge University Press, New York.
- Sun, R. & Naveh, I. (2004). Simulating organizational decision-making using a cognitively realistic agent model. *Journal of Artificial Societies and Social Simulation*, 7(3).
- Sutton, R. S. (1988). Learning to predict by the methods of temporal differences. *Machine Learning*, 3(1):9–44.

- Swingler, K. (2012). On the capacity of Hopfield neural networks as EDAs for solving combinatorial optimisation problems. In *Proceedings of the International Joint Conference on Computational Intelligence (IJCCI)*, pages 152–157.
- Sylvester, J. C. & Reggia, J. A. (2013). The neural executive: Can gated attractor networks account for cognitive control? In *Proceedings of the Annual Meeting of the International Association for Computing & Philosophy*, volume 37, College Park, MD.
- Sylvester, J. C. & Reggia, J. A. (2014). Engineering neural systems for cognitive control. *Submitted*.
- Sylvester, J. C., Reggia, J. A., & Weems, S. A. (2011). Cognitive control as a gated cortical net. In *Proceedings of the 2nd International Conference on Biologically Inspired Cognitive Architectures*, pages 371–376. IOS Press.
- Sylvester, J. C., Reggia, J. A., Weems, S. A., & Bunting, M. F. (2010). A temporally asymmetric Hebbian network for sequential working memory. In Salvucci, D. D. & Gunzelmann, G., editors, *Proceedings of the 10th International Conference on Cognitive Modeling*, pages 241–246.
- Sylvester, J. C., Reggia, J. A., Weems, S. A., & Bunting, M. F. (2013). Controlling working memory with learned instructions. *Neural Networks*, 41(0):23–38.
- Tang, K. W. & Kak, S. (2002). Fast Classification Networks For Signal Processing. *Circuits, Systems, and Signal Processing*, 21(2):207–224.
- Tanji, J., Shima, K., & Mushiake, H. (2007). Concept-based behavioral planning and the lateral prefrontal cortex. *Trends in Cognitive Sciences*, 11:528–534.
- Thompson, T. W., Waskom, M. L., Garel, K.-L. A., Cardenas-Iniguez, C., Reynolds, G. O., Winter, R., Chang, P., Pollard, K., Lala, N., Alvarez, G. A., & Gabrieli, J. D. E. (2013). Failure of working memory training to enhance cognition or intelligence. *PLoS One*, 8(5):e63614.
- Tieleman, T. & Hinton, G. E. (2009). Using fast weights to improve persistent contrastive divergence. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 1033–1040. ACM.
- Tolman, E. C. & Honzik, C. H. (1930). “Insight” in rats. *University of California Publications in Psychology*, 4(4):215–232.
- Touretzky, D. S. (1990). BoltzCONS: Dynamic symbol structures in a connectionist network. *Artificial Intelligence*, 46(1–2):5–46.
- Touretzky, D. S. & Hinton, G. E. (1988). A distributed connectionist production system. *Cognitive Science*, 12(3):423–466.
- Townsend, J., Keedwell, E., & Galton, A. (2014). Artificial development of biologically plausible neural-symbolic networks. *Cognitive Computation*, 6(1):18–34.
- Trullier, O., Wiener, S., Berthoz, A., & Meyer, J.-A. (1997). Biologically based artificial navigation systems: Review and prospects. *Progress in Neurobiology*,

- 51(5):483–544.
- Tsuda, I. (2001). Toward an interpretation of dynamic neural activity in terms of chaotic dynamical systems. *The Behavioral and Brain Sciences*, 24(5):793–810.
- Ungerleider, L. G. & Haxby, J. V. (1994). ‘What’ and ‘where’ in the human brain. *Current Opinion in Neurobiology*, 4(2):157–165.
- Usatenko, O. V., Melnik, S. S., Apostolov, S. S., Makarov, N. M., & Krokhin, A. A. (2014). Iterative method for generating correlated binary sequences. *arXiv:1406.2656v1*.
- van Essen, D. (2005). Corticocortical and thalamocortical information flow in the primate visual system. *Progress in Brain Research*, 149:173–185.
- van Essen, D., Anderson, C., & Felleman, D. (1992). Information processing in the primate visual systems. *Science*, 255:419–423.
- van Veen, V. & Carter, C. S. (2006). Conflict and cognitive control in the brain. *Current Directions in Psychological Science*, 15(5):237–240.
- Varma, S. & Just, M. A. (2006). 4CAPS: An adaptive architecture for human information processing. In *AAAI Spring Symposium: Between a Rock and a Hard Place: Cognitive Science Principles Meet AI-Hard Problems*, pages 91–96.
- Verduzco-Flores, S., Ermentrout, B., & Bodner, M. (2012). Modeling neuropathologies as disruption of normal sequence generation in working memory networks. *Neural Networks*, 27:21–31.
- Walsh, M. M. & Anderson, J. R. (2010). Neural correlates of temporal credit assignment. In *Proceedings of the 10th International Conference on Cognitive Modeling*, pages 265–270.
- Wang, D. (1995). Emergent synchrony in locally coupled neural oscillators. *IEEE Transactions Neural Networks*, 6:941–7.
- Watter, S., Geffen, G. M., & Geffen, L. B. (2001). The n -back as a dual-task: P300 morphology under divided attention. *Psychophysiology*, 38(6):998–1003.
- Weems, S. & Reggia, J. A. (2006). Simulating single word processing in the classic aphasia syndromes based on the Wernicke-Lichtheim-Geschwind theory. *Brain and Language*, 98:291–309.
- Weems, S. A., Winder, R. K., Bunting, M. F., & Reggia, J. A. (2009). Running memory span: A comparison of behavioral capacity limits with those of an attractor neural network. *Cognitive Systems Research*, 10(2):161–171.
- Winder, R., Cortez, C., Reggia, J. A., & Tagamets, M. (2007). Functional connectivity in fMRI: A modeling approach for estimation and for relating to local circuits. *NeuroImage*, 34:1093–1107.
- Winder, R. K., Reggia, J. A., Weems, S. A., & Bunting, M. F. (2009). An oscillatory Hebbian network model of short-term memory. *Neural Computation*, 21(3):741–761.

- Womelsdorf, T. & Fries, P. (2009). Selective attention through selective neuronal synchronization. In Gazzaniga, M., editor, *The Cognitive Neurosciences*, pages 289–302. MIT Press.
- Zhang, L., Tao, H., Holt, C., Harris, W., & Poo, M. (1998). A critical window for cooperation and competition among developing retinotectal synapses. *Nature*, 395:37–44.
- Zipser, D. (1991). Recurrent network model of the neural mechanism of short-term active memory. *Neural Computation*, 3(2):179–193.
- Zipser, D., Kehoe, B., Littlewort, G., & Fuster, J. M. (1993). A spiking network model of short-term active memory. *Journal of Neuroscience*, 13(8):3406–20.