Controlling Working Memory with Learned Instructions

J.C. Sylvester[a,b,*], J.A. Reggia[a,b], S.A. Weems[b], M.F. Bunting[b]

[a] *Department of Computer Science, University of Maryland*
A.V. Williams Building, College Park, MD 20742
[b] *Center for Advanced Study of Language, University of Maryland*
7005 52nd Avenue, College Park, MD 20742

Abstract

Many neural network models of cognition rely heavily on the modeler for control over aspects of model behavior, such as when to learn and whether an item is judged to be present in memory. Developing neurocomputational methods that allow these cognitive control mechanisms to be performed autonomously has proven to be surprisingly difficult. Here we present a general purpose framework called GALIS that we believe is amenable to developing a broad range of cognitive control models. Models built using GALIS consist of a network of interacting "regions" inspired by the organization of primate cerebral cortex. Each region is an attractor network capable of learning temporal sequences, and the individual regions not only exchange task-specific information with each other, but also gate one another's functions and interactions. As a result, GALIS models can learn both task-specific content and also the necessary cognitive control procedures (instructions) needed to perform a task in the first place. As an initial test of this approach, we use GALIS to implement a model that is trained simultaneously to perform five versions of the n-Back task. Not only does the resulting n-Back model function correctly, determining when to learn or remove items in working memory, but its accuracy and response times correlate strongly with those of human subjects performing the same task. The n-Back model also makes testable predictions about how human accuracy would be affected by intra-trial changes in n's value. We conclude that GALIS opens a potentially effective pathway towards developing a range of cognitive control models with improved autonomy.

Keywords: cognitive control, working memory, n-back, Hopfield attractor networks, adaptive gating, temporally asymmetric Hebbian learning

1. Introduction

"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 (Roy, 2008). However, developing such architectures has proven to be surprisingly challenging. Neural systems currently excel at problems that make use of strengths like pattern matching and incremental learning (Omlin & Giles, 2000), 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 (pattern matching, 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

*Corresponding author. Tel: 301-922-2576. Fax: 301-405-6707

Email addresses: jared@cs.umd.edu (J.C. Sylvester), reggia@cs.umd.edu (J.A. Reggia), sweems@casl.udm.edu (S.A. Weems), bunting@casl.umd.edu (M.F. Bunting)

URL: http://www.cs.umd.edu/~jared/ (J.C. Sylvester)

computation has led, among other things, to the development of pioneering neural models that explicitly incorporate aspects of cognitive control, such as for managing working memory (O'Reilly & Frank, 2006) and for planning solutions to the Towers of London problem (Dehaene & Changeux, 1997).

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 (Frank et al., 2001), 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 & 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.

In this paper, we describe an approach to building models of cognitive control for working memory tasks which we call 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. Our goal is not to provide a veridical model of the neurobiology underlying human cognitive control; rather, we are taking inspiration from cerebral cortex to create a general computational framework that can be used effectively to create a broad range of neural architectures for specific tasks. In the following, we discuss some past related work (Section 2) before describing our vision for the GALIS framework in general terms (Section 3). The remainder of the paper focuses on a first test for this framework. In particular, we use GALIS to develop a system for the specific task of controlling working memory during performance of n-Back tasks where the parameter n can change dynamically during task performance (Section 4). We show that this n-Back system not only works, but that at

least on some measures, it also performs similarly to human behavior, and it makes testable predictions that could falsify the model (Section 5). Finally, we discuss the current state of the GALIS framework and future directions for its expansion (Section 6).

2. Past Related Work

Numerous neural network models of working memory exist (e.g., Burgess & Hitch, 1999; Botvinick & Plaut, 2006; Blum & Abbott, 1996; Kaplan et al., 2006; Page & Norris, 1998; Pascanu & Jaeger, 2011; Ponzi, 2008; Verduzco-Flores et al., 2012). However, these models often have no endogenous control at all. For example, in approaches such as OSCAR (Brown et al., 2000), TODAM (Li & Lewandowsky, 1993), SOB (Farrell & Lewandowsky, 2002) and Itinerant Attractors (Hoshino et al., 1997), all decisions about when to update weights or assessments about whether an item is in memory are made externally by a human. Of models with endogenous control, many are quite specific to their given tasks. Overcoming this specificity has recently become a goal of multiple researchers, for instance, Frank et al. (2007).

One notable prior neural network model which does have generality as a goal is that of Rougier et al. (2005). The authors present a model which is capable of performing a variety of tasks in which the key objective is identifying the single currently relevant feature, such as variations to the Wisconsin Card Sort. Using an iterative learning process, the network is able to discover internal, rule-like representations which allow it to isolate single features. Importantly, no architectural changes are required to shift between this set of tasks, and concurrent training on some tasks improves the ability of the model to generalize to novel stimuli in other tasks. Unlike with the GALIS framework, Rougier et al. frame their tasks as a form of pattern classification, using an iterative training regime to learn input-output mappings. In contrast, the system presented here uses one-shot learning, and frames each task as a procedure to be executed.

Chatham et al. (2011) use the PBWM architecture to build a model capable of executing 2- and 3-back tasks at human performance levels. Their model differs in a number of ways from that presented here. One major difference is that Chatham et al. present their model with input consisting of both the letter to be remembered *and* that letter's serial order. Furthermore, the serial order is given to the model as a periodic coding which aligns with the value of n. So while the model described here 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 and 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 we 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 the period (so that, for instance, G3 displaces D3 in the example above), while we use weight decay and Hebbian unlearning to minimize interference from older, irrelevant stimuli.

A number of past models must confront related problems with scaling. For example, some working memory models require a separate layer, or set of layers, for every item which could be stored in memory (e.g., Jones & Polk, 2002; Stewart et al., 2011). Others require extra layers for every rule or action (e.g., Simen et al., 2010; Stewart & Eliasmith, 2011). This presents both a computational problem (because such models scale poorly to more complex tasks and environments), as well as a plausibility problem (because it contradicts what is known about the pervasive re-use of biological neural circuits).

3. The GALIS Framework

GALIS is intended to be a general neurocomputational framework for modeling learned cognitive control of working memory tasks. By working memory, we mean the ability to monitor, hold and manipulate information that is needed for performing tasks over the short term (Durstewitz et al., 2000). Working memory is of 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.

While GALIS is not intended to be a veridical 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 & 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, 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 models of cognitive control, 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 full autonomy.

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., Jones & Polk, 2002; Farrell & Lewandowsky, 2002; Hoshino et al., 1997), but usually they are limited to dealing with only fixed-point attractors. However, if working memory is to accomodate procedural information that supports cognitive control, it must also be able to store attractors that are 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 & 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., 2010, 2011).

The third hypothesis is that each cortical region can not only exchange information with other cortical regions in the form of activity patterns, 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, and 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 & Guillery, 2006; 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 & 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 & Frank, 2006). In contrast, in the GALIS framework the details of implementing gating actions via complex subcortical circuity, 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 cerebral cortex, our approach to 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 and whether to learn and unlearn stimuli. At the present time, we are also using distributed rather than local representations and one-shot Hebbian learning rather than error-backpropagation, both of which are intended to increase the biological plausibility of the GALIS approach.

Is the GALIS framework an effective approach to creating models of cognitive control? The claim that it is effective is falsifiable: if one cannot successfully construct a model for a specific working memory task within the framework, that would indicate that the three core hypotheses that it is based upon are insufficient. As a first step in testing the adequacy of this framework, we have used it to implement a specific model that performs multiple versions of the n-Back task. This model is capable of determining when to learn or remove items in working memory, as well as of assessing the contents of its memory and comparing them to external stimuli. Control is exercised endogenously, and behaviors are based on the contents of an instruction memory that learns and stores the actions necessary for the task. The next two sections of this paper describe this n-Back model and the results obtained with it. The work reported here greatly extends a preliminary report done as a pilot study with a much simpler toy "store and retrieve" memory problem (Sylvester et al., 2011). It establishes for the first time that GALIS models can learn to perform a more difficult, real-world working memory learning task that is widely used in cognitive psychology.

4. Methods

The GALIS model of n-Back works on tasks in the context of a sequential stream of visual inputs. 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. For example, in a 3-back task the bold letters in the following sequence would be considered matches: V H Z **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., Watter et al., 2001; Schmidt et al., 2009), correlated with general intelligence (Jaeggi et al., 2008), and used for training to improve working memory capacity (Jaeggi et al., 2010). The remainder of this section gives an overview of each component of the GALIS model for n-Back, describes how it operates to process stimuli, and covers details of the internal structure of the control module. Further details are given in the Appendix.

4.1. Top level architecture for n-Back tasks

The GALIS model for performing n-Back tasks consists of several interacting regions, as seen in Figure 1. They are the visual input layer, the n-input

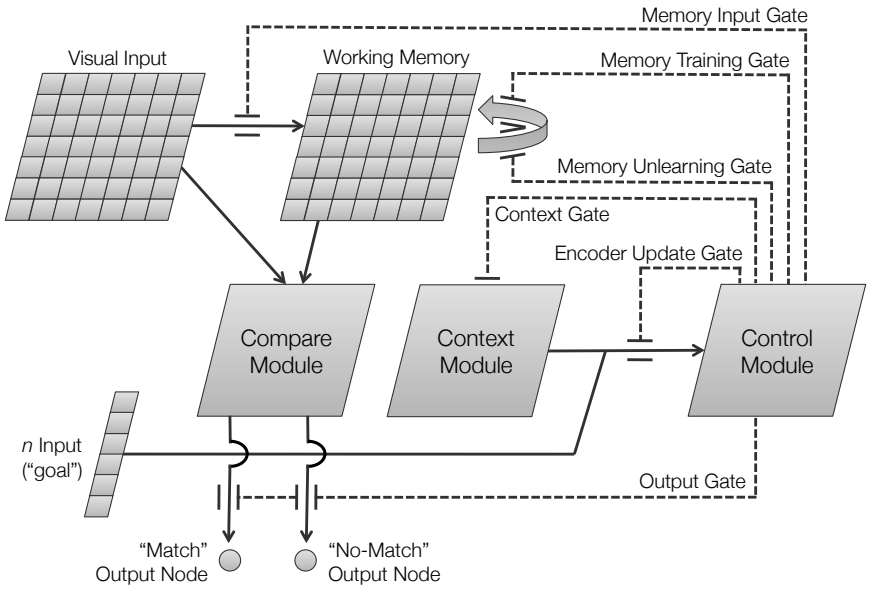

Figure 1. The GALIS model as used for the n-Back task. Thin, solid arrows denote one-to-one connections. The working memory layer is fully recurrently connected (broad arrow). Dotted lines are the outputs of the control module. Note that the number of boxes pictured in each layer is an approximation only, and does not faithfully represent the number of nodes used in the model.

layer, the output nodes, the memory layer, the compare module, the context module, and the control module.

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 & Koechlin, 2010), and which we intend to add in a future extended version of GALIS. In the case of modeling the n-Back task, this n-input layer encodes which particular version of n-Back the model should currently be perform-

ing (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, we plan on extending GALIS to use the same input mechanism to enable it 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 Hopfield network forming an auto-associative memory (Hopfield, 1982). 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 (Reggia et al., 2009; Winder et al., 2009) and temporally asymmetric weights, which enable it to recall a temporal sequence of stored patterns in a specified order rather than randomly (Sylvester et al., 2010, 2011). The working memory layer is the same size as the visual input layer, and its nodes are bipolar valued. The working memory layer is treated in more depth in Sections 4.2 and A.1.

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 Appendix Section A.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 Appendix Section A.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. The core of the control module is a second discrete Hopfield 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, 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 2). 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 founds in Sections A.2 and A.3.

[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.

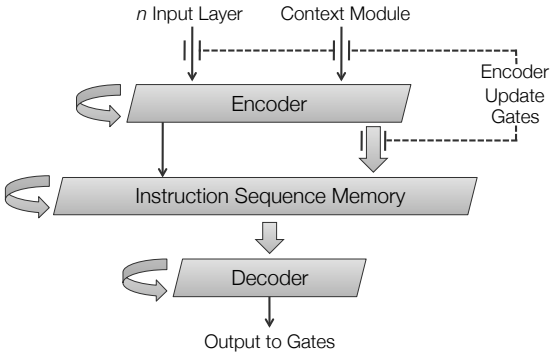

Figure 2. Slightly expanded control module showing its gated input encoder and output decoder which function as heteroassociative memories.

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 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 1. 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

$$g(t) = k_g\, g(t-1) + s(t) \tag{1}$$

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.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.

4.2. Working Memory

The working memory layer is a discrete Hopfield network incorporating dynamic thresholds and temporally asymmetric weights that permit it to process temporal sequences as described in 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. Appendix Section A.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. 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 Eq. 1. 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.

4.4. Controller Functionality

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 1; 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

Table 1. Instruction sequences learned by the control module's ISM.

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

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 Appendix Sect. A.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 3 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.

In Figure 3(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 1 ("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 3(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 3(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 3(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 3(e) the gates remain unchanged as the working memory recalls the preceding item again. In Figure 3(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 3(f) is done.

5. 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.

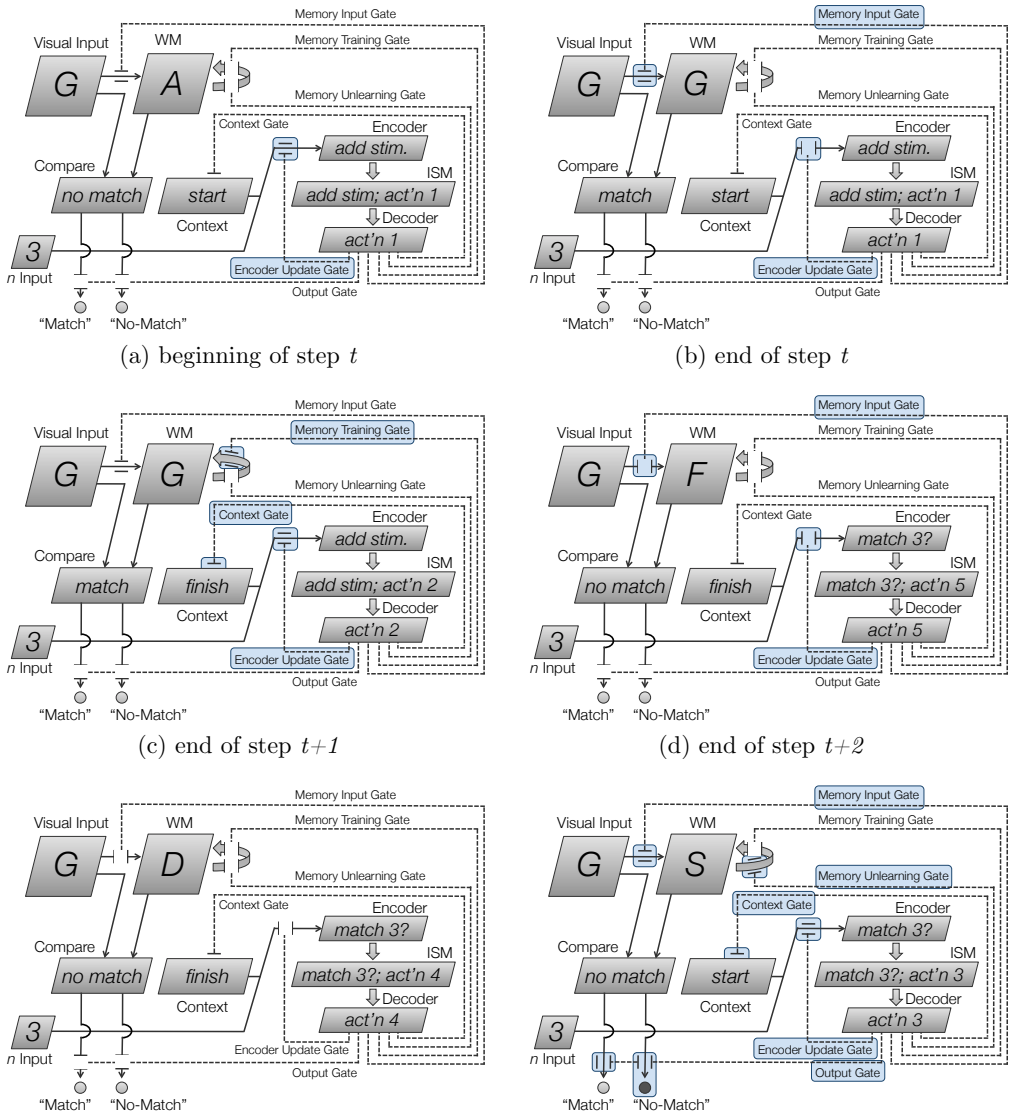

Figure 3. Step-by-step operation of the n-Back model to process one input stimulus G. (See text for details.)

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 & 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.

A D J **A** E F D K J C **K** A H F **A H** G D F **G D** K A C **K** H **C** A G J **A G** K

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, 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 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.[2]

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

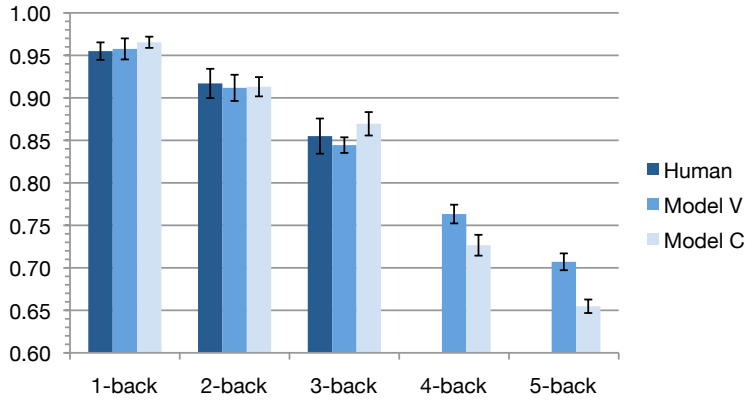

Figure 4. Accuracy for human subjects and the computational model for all five versions of n-Back. Human results were not reported for $n=4$ and $n=5$. Error bars represent the standard error of the mean. Model V used different working memory decay rates (k_{WM}) for each n, while Model C did not vary decay.

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 (Reggia et al., 2009; Sylvester et al., 2010). A lower decay 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 they are attempting to remember (Altmann & Gray, 2002; Winder et al., 2009). The additional degree of freedom in Model V may account for it's 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 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 $k_{\mathrm{WM}} = .2625$ in Model C therefore disproportionally 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 1, 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 5. The GALIS n-Back model's response time correlates well with the human response time data, with $\mathrm{R}^2 = 0.9888$. 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,

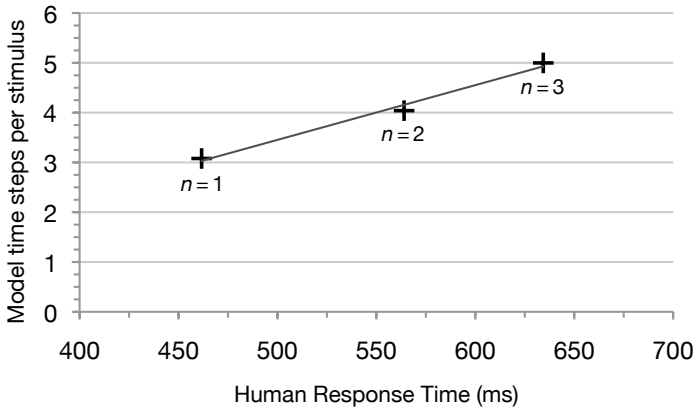

Figure 5. Correlation between human response time per stimulus and the average number of steps the model takes per stimulus. The trend line is defined by $y = 0.011x - 2.042$, with $\mathrm{R}^2 = 0.9888$.

the value of the n-input layer was set to $n_2 \neq n_1$.[3] 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 6 shows some representative results when mid-trial changes in n occur. Each graph shows the accuracy at each position in the input sequence, 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

[3]For all trials in these experiments, $k_{\mathrm{ISM}} = -.3$ and $k_{\mathrm{WM}} = .225$. (See Appendix.)

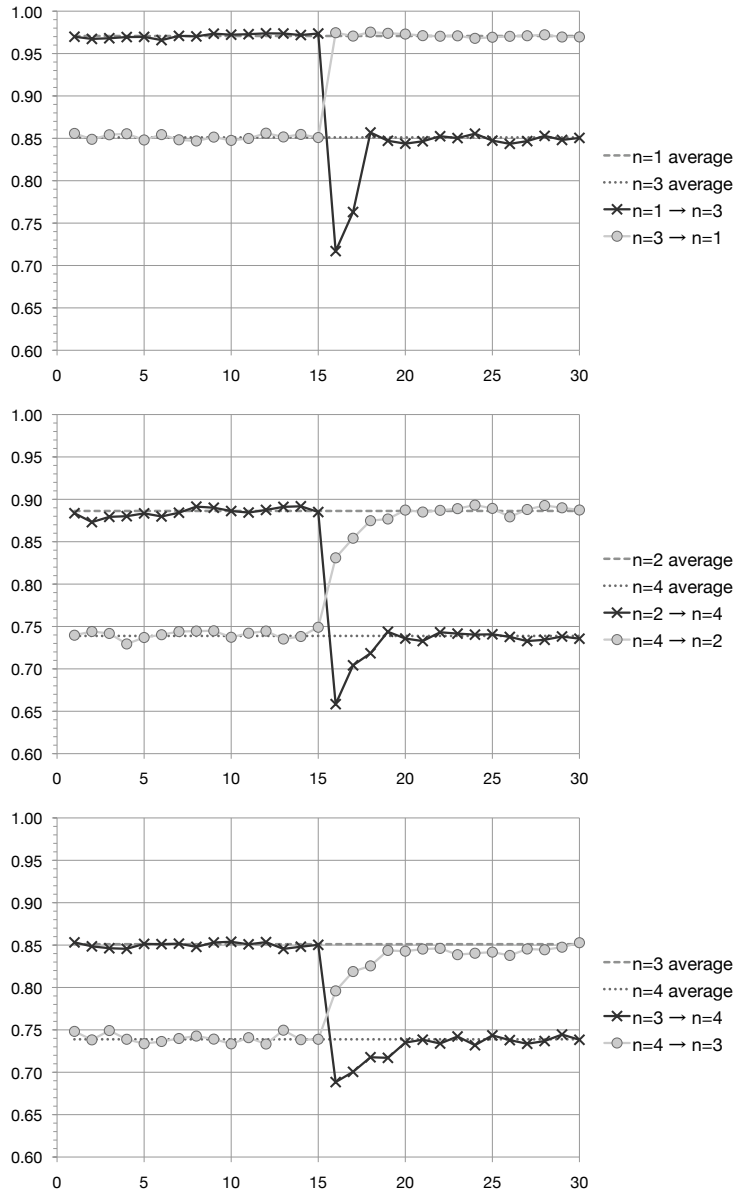

Figure 6. Accuracy when changing the value of n during a trial. The horizontal axis is the serial position within each sequence (following the preparatory period), and the vertical axis is the average accuracy of responses to that stimulus across all trials. Values of n were switched beginning with the 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 $n = 4$.

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-trail 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 1) should occur in that order for each stimulus. We then computed the Levenshtein distance[4] 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

[4]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.

properly identified, or a failure to advance to the previously trained item. (It is possible that the 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 & Valabregue, 1999).

The proportion of runs having control errors can be seen in Fig. 7(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. 7(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. 7(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 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

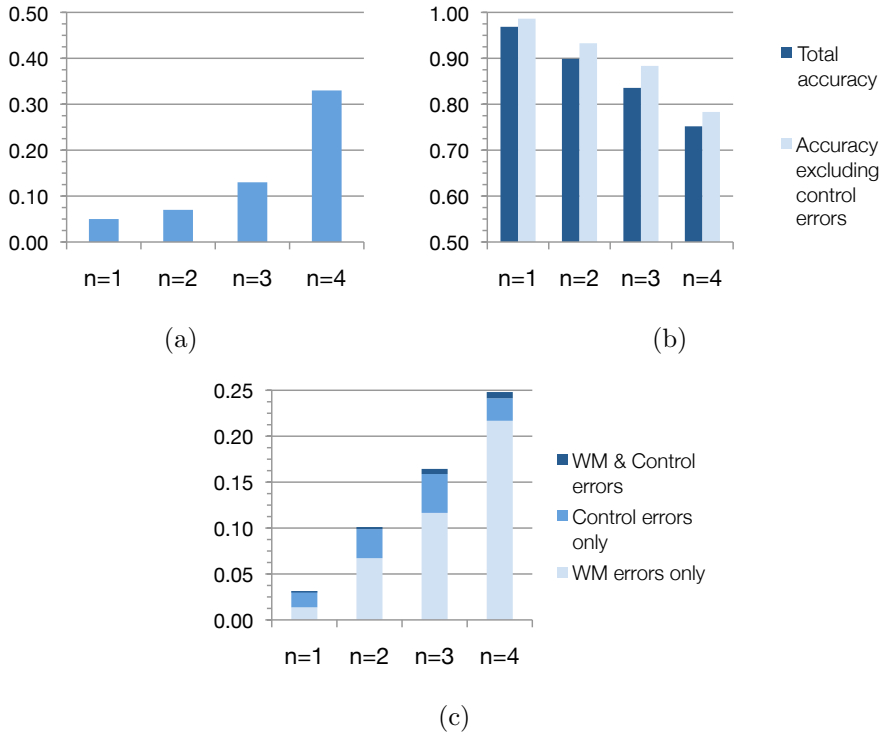

Figure 7. (a) The proportion of runs in which the control module made one or more errors for $n \in \{1, 2, 3, 4\}$. (b) The accuracy of model outputs for all runs (darker bars), and for those runs where no control errors were made (lighter bars). (c) The proportion of model responses which were incorrect due to malfunctions in the working memory, controller, or both.

semi-orthogonal patterns to represent each action, or ensuring there is a minimum Hamming distance between internal representations.

6. Discussion

6.1. 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 & 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. We are 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 we would expect to see the patterns evident in Figure 6.

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 behavior in the model. Additionally, there are only a few major architectural differences between this model and a previously implemented model of a much simpler task (Sylvester et al., 2011). These consist primarily

of adjusting output — and to a lesser degree, input — systems 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 will 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 this and our prior work is also a dedicated piece of architecture, hand-designed by the modelers. This does decrease the generality of the model, but we 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 will generally be useful for a spectrum of future tasks.

6.2. 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, we predict that changing the information learned by the control module will enable GALIS models to be adapted to other widely used serial recall tasks such as Running Memory Span or Block Span. In other words, we believe that building models of other similar tasks will require mostly changes to the neural software and only minimal changes to the neural hardware. To test this claim, work is currently ongoing to expand GALIS from multiple versions of n-Back to different serial memory and continuous performance tasks. Tasks currently being investigated include the AX-CPT and 1-2 AX-CPT variant (Carter et al., 1998; O'Reilly & Frank, 2006) as well as the Wisconsin Card Sort Test (Rougier et al., 2005) and the Wisconsin Delayed-Match-to-Sample task (Stemme et al., 2007).

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 & Seamans, 2008). Attractors dominated by D1-type dynamics have deep basins, aiding robustness of working memory but increasing preseveration, 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.

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 & Polk, 2009). This is reminiscent of the way resonant states in ART networks balance characteristics of distributed and discrete representations (Carpenter & Grossberg, 2003; Grossberg, 2000), which leads us to consider the potential usefulness of adapting some of ART's learning methods to improve GALIS's attractor networks in the future. We 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.

Of course, the GALIS framework currently has some limitations, and will evolve in its details as we explore its use for additional tasks. These limitations include requiring learned instruction sequences to be determined by the modeler. Despite this issue, we 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, we 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. We are 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 we 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. 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 & Lewandowsky, 2002; Koene & Hasselmo, 2007) which we are exploring to resolve this issue, in addition to looking into other neural approaches to serial memory (e.g., Botvinick & Plaut, 2006; Kremer, 2001; Monner & Reggia, 2012).

Further, the experiments described here were performed without lure stimuli, which are a heavily studied aspect of n-Back performance. Our goal with the present paper was not to provide an exhaustive elucidation of how humans solve n-Back, but simply to offer demonstration that the techniques of GALIS are valuable for solving the same sorts of problems as humans do. Having done so, we now plan a more thorough study of how the GALIS n-Back model compares to humans, including on performance on lure trials, the affects of training, and other aspects of n-Back in the near future.

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 & Braver, 2005; MacDonald et al., 2000). An important direction for future research will be to further bring GALIS into alignment with known neuroscientific data.

Acknowledgements
Supported by funding from the US Government.

Appendix. Model Details

A.1. Working Memory

The working memory layer is based on the temporally asymmetric attractor approach developed in Sylvester et al. (2010). 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:

$$w_{ij}(t) = (1 - k_{\mathrm{WM}})\, w_{ij}(t-1) + \frac{1}{N}\, a_i(t)\, a_j(t)\, (1 - \delta_{ij}) \tag{A.1}$$

Here k_{WM} is the decay rate ($0 \leq k_{\mathrm{WM}} < 1$) 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

$$v_{ij}(t) = (1 - k_{\mathrm{WM}})\, v_{ij}(t-1) + \frac{1}{N}\, a_i(t)\, a_j(t-1) \tag{A.2}$$

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:

$$h_i(t) = \sum_j v_{ij}\, a_j(t-1) - \theta_i(t) + 2\, g_{in}(t)\, \mathrm{in}_i(t) \tag{A.3}$$

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. Specifically, at every time step, θ_i decays according to $\theta_i(t+1) = (1 - k_\theta)\,\theta_i(t)$ and in any time step in which the state of node i is unchanged from the previous time step a factor of $k_w\,a_i(t)$ is also added to $\theta_i(t+1)$. 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 Eq. A.4.

$$a_i(t) = \begin{cases} +1 & h_i(t-1) > 0 \\ a_i(t-1) & h_i(t-1) = 0 \\ -1 & h_i(t-1) < 0 \end{cases} \tag{A.4}$$

The $h_i(t) = 0$ case is used to prevent the nodes from being biased towards either turning on or off when their inputs are perfectly balanced. (This situation is extremely rare.)

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:

$$f_i(t) = \sum_j w_{ij}\,a_j(t) - \theta_i(t) + 2\,g_{\text{in}}(t)\,in_i(t) \tag{A.5}$$

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 A.4 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:

$$w_{ij}(t) = w_{ij}(t-1) - \frac{1}{2^{n-1}} \cdot \frac{1}{N}\,a_i(t)\,a_j(t)\,(1 - \delta_{ij}) \tag{A.6}$$

$$v_{ij}(t) = v_{ij}(t-1) - \frac{1}{2^{n-1}} \cdot \frac{1}{N}\,a_i(t)\,\operatorname{sgn}\left(\sum_{l=1}^{N} v_{lj} a_l(t)\right) \tag{A.7}$$

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 Eq. A.2 has been replaced in Eq. A.7 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.

A.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 1 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 1) 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 1 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.)

A.3. Controller Architecture

A diagram of the control module's internal structure can be seen in Figure A.8. 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 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.

A.3.1. 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 (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 WM 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 1.

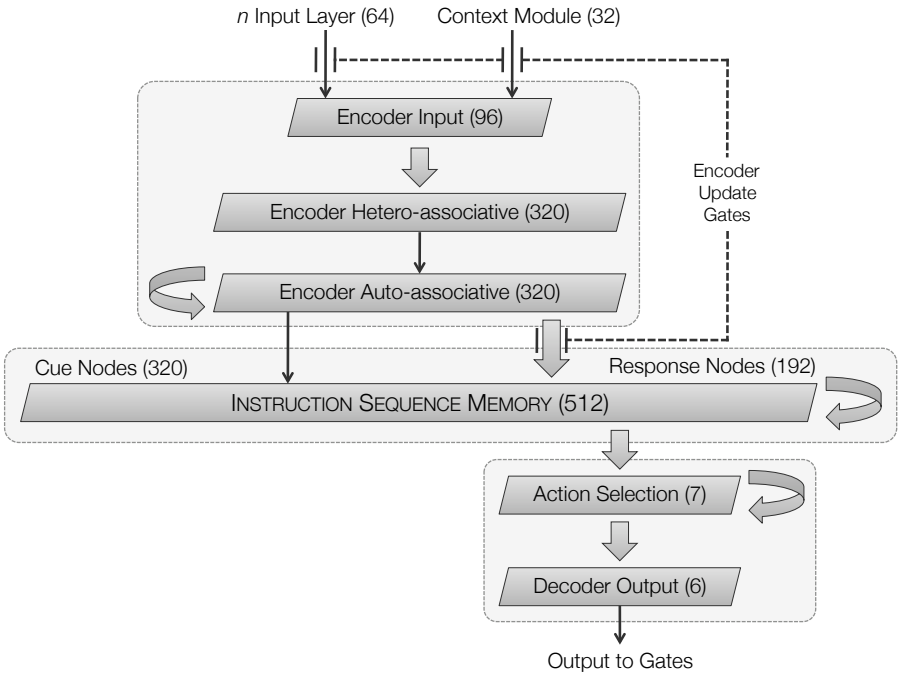

Figure A.8. The control module. Thick arrows denote fully connected layers, while thin arrows denote one-to-one connections. Following the name of each layer is the number of nodes it contains in the example presented in this paper. One-to-one connections from the Encoder Auto-associative layer terminate on the cue nodes of the instruction sequence memory, while full connections terminate on the response nodes.

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 1).

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

$$e_i = \begin{cases} a_{\text{enc}_i} & i \in \{\textsc{Cue}\} \\ g_{\text{ctrl}} \sum_{k \in \text{enc}} u_{ik} a_{\text{enc}_k} & i \in \{\textsc{Response}\} \end{cases} \tag{A.8}$$

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, 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:

$$w_{ij}(\tau) = (1 - k_{\text{ISM}})\, w_{ij}(\tau{-}1) + \frac{1}{N}\, a_i(\tau)\, a_j(\tau)\, (1 - \delta_{ij}) \tag{A.9}$$

$$v_{ij}(\tau) = (1 - k_{\text{ISM}})\, v_{ij}(\tau{-}1) + \frac{1}{N}\, a_i(\tau{-}1)\, a_j(\tau) \tag{A.10}$$

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 1. 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. A.3 and A.5 are redefined as

follows to accommodate the differences in the external input, given in Eq. A.8.

$$h_i(t) = \sum_j v_{ij}\, a_j(t-1) - \theta_i(t) + e_i(t) \tag{A.11}$$

$$f_i(t) = \sum_j w_{ij}\, a_j(t) - \theta_i(t) + e_i(t) \tag{A.12}$$

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.

A.3.2. 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 in completed. The encoder-to-response-node connections are mediated by the same gate value, since they should 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. A.8). 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 & 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 & Eliasmith, 2010), among others). Attractor networks have been used for this purpose before, including Lewandowsky (1999) and Kesner et al. (2000).

A.3.3. 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. A.8). The first, called the action selection layer, has seven binary nodes — as many as there are response patterns.[5] 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 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

[5]Using one node per action is a violation of our commitment to using distributed representations. While we are exploring alternate arrangements for the future, 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., 2001; 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.

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 $(1, -0.25, 0, 0, 0, 0, 0)$.

A.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.

A.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.

References

Altmann, E. M., & Gray, W. D. (2002). Forgetting to remember: The functional relationship of decay and interference. *Psychological Science*, *13*, 27–33.

Amos, A. (2000). A computational model of information processing in the frontal cortex and basal ganglia. *Journal of Cognitive Neuroscience*, *12*, 505–19.

Baddeley, A. D. (2007). *Working memory, thought, and action*. Number 45 in Oxford Psychology Series. Oxford University Press.

Blum, K. I., & Abbott, L. F. (1996). A model of spatial map formation in the hippocampus of the rat. *Neural Computation*, *8*, 85–93.

Botvinick, M. M., & Plaut, D. C. (2006). Short-term memory for serial order: A recurrent neural network model. *Psychological Review*, *113*, 201–233.

Bressler, S., & Menon, V. (2010). Large-scale brain networks in cognition. *Trends in Cognitive Sciences*, *14*, 277–290.

Brown, G. D., Preece, T., & Hulme, C. (2000). Oscillator-based memory for serial order. *Psychological Review*, *107*, 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.

Burgess, N., & Hitch, G. J. (1999). Memory for serial order: A network model of the phonological loop and its timing. *Psychological Review*, *106*, 551–581.

Carpenter, G. A., & Grossberg, S. (2003). Adaptive resonance theory. In M. A. Arbib (Ed.), *The Handbook of Brain Theory and Neural Networks* (pp. 87–90). Cambridge, MA: MIT Press.

Carter, C. S., Braver, T. S., Barch, D. M., Botvinick, M. M., Noll, D. C., & Cohen, J. D. (1998). Anterior cingulate cortex, error detection, and the online monitoring of performance. *Science*, *280*, 747–9.

Charron, S., & Koechlin, E. (2010). Divided representation of concurrent goals in the human frontal lobes. *Science*, *328*, 360–363.

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*, 3598–619.

Choo, F.-X., & Eliasmith, C. (2010). Spiking neuron model of serial-order recall. In R. Cattrambone, & S. Ohlsson (Eds.), *32nd Ann. Conf. Cognitive Science Society*. Portland, OR.

Cowan, N. (1999). An embedded-processes model of working memory. In A. Miyake, & P. Shah (Eds.), *Models of working memory: Mechanisms of active maintenance and executive control* (pp. 62–101). Cambridge University Press.

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*, 42–100.

Dehaene, S., & Changeux, J.-P. (1997). A hierarchical neuronal network for planning behavior. *Proc. Nat'l Acad. Sci. USA*, *94*, 13293–8.

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*, 739–49.

Durstewitz, D., Seamans, J. K., & Sejnowski, T. J. (2000). Neurocomputational models of working memory. *Nature Neuroscience*, *3*, 1184–1191.

Engle, R. W., Conway, A. R. A., Tuholski, S. W., & Shisler, R. J. (1995). A resource account of inhibition. *Psychological Science*, *6*, 122–125.

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., & D., F. (1992). Information processing in the primate visual systems. *Science*, *255*, 419–423.

Farrell, S., & Lewandowsky, S. (2002). An endogenous distributed model of ordering in serial recall. *Psychonomic Bulletin & Review*, *9*, 59–79.

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.

Frank, M. J., Samanta, J., Moustafa, A. A., & Sherman, S. J. (2007). Hold your horses: Impulsivity, deep brain stimulation, and medication in Parkinsonism. *Science*, *318*, 1309–12.

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.

Grossberg, S. (2000). Adaptive resonance theory. In L. Nadel (Ed.), *Encyclopedia of Cognitive Science*. Wiley.

Gurney, K., Prescott, T. J., & Redgrave, P. (2001). A computational model of action selection in the basal ganglia, I: A new functional anatomy. *Biological Cybernetics*, *84*, 401–410.

Harbison, J. I., Atkins, S. M., & Dougherty, M. R. (2011). Performance gains in an adaptive n-back working memory training task. In *Proc. 50th Ann. Mtg. Psychonomic Society* (pp. 120–125).

Hockey, A., & Geffen, G. (2004). The concurrent validity and test–retest reliability of a visuospatial working memory task. *Intelligence*, *32*, 591–605.

Hopfield, J. J. (1982). Neural networks and physical systems with emergent collective computational abilities. *Proc. Nat'l Acad. Sci. USA*, *79*, 2554–2558.

Hopfield, J. J., Feinstein, D. I., & Palmer, R. G. (1983). 'Unlearning' has a stabilizing effect in collective memories. *Nature*, *304*, 158–159.

Horn, D., & Usher, M. (1992). Oscillatory model of short term memory. In J. E. Moody, S. J. Hanson, & R. P. Lippmann (Eds.), *Advances in Neural Infromation and Processing Systems 4* (pp. 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*, 1375–1390.

Jaeggi, S. M., Buschkuehl, M., Jonides, J., & Perrig, W. J. (2008). Improving fluid intelligence with training on working memory. *Proc. Nat'l Acad. Sci. USA*, *105*, 6829–6833.

Jaeggi, S. M., Studer-Luethi, B., Buschkuehl, 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*, 625–635.

Jones, M., & Polk, T. A. (2002). An attractor network model of serial recall. *Cognitive Systems Research*, *3*, 45–55.

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*, 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*, 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*, 260–5.

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.

Korsten, N. J. H., Fragopanagos, N., Hartley, M., Taylor, N., & Taylor, J. G. (2006). Attention as a controller. *Neural Networks*, *19*, 1408–1421.

Kremer, S. C. (2001). Spatiotemporal connectionist networks: A taxonomy and review. *Neural Computation*, *13*, 249–306.

Lewandowsky, S. (1999). Redintegration and response suppression in serial recall: A dynamic network model. *Int'l Jrnl of Psychology*, *34*, 434–446.

Lewandowsky, S., & Farrell, S. (2003). Computational models of working memory. In L. Nadel (Ed.), *Encyclopedia of Cognitive Science*. Macmillan Reference.

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*, 895–908.

Long, D. L., Parks, R. W., & Levine, D. S. (1998). An introduction to neural network modeling: Merits, limitations, and controversies. In R. W. Parks, D. S. Levine, & D. L. Long (Eds.), *Fundamentals of Neural Network Modeling* (pp. 3–32). Cambridge, MA: MIT Press.

Ma, J. (1999). The asymptotic memory capacity of the generalized Hopfield network. *Neural Networks*, *12*, 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.

Marcus, G. F. (2001). *The Algebraic Mind: Integrating Connectionism and Cognitive Science*. Cambridge, MA: MIT Press.

McEliece, R., Posner, E., Rodemich, E., & Venkatesh, S. (1987). The capacity of the Hopfield associative memory. *IEEE Trans. Information Theory*, *33*, 461–482.

Monner, D., & Reggia, J. A. (2012). A generalized LSTM-like training algorithm for second-order recurrent neural networks. *Neural Networks*, *25*, 70–83.

Navarro, G. (2001). A guided tour to approximate string matching. *ACM Computing Surveys*, *33*, 31–88.

Omlin, C. W., & Giles, L. (2000). Symbolic knowledge representation in recurrent neural networks: Insights from theoretical models of computation. In I. Cloete, & J. M. Zuruda (Eds.), *Knowledge Based Neurocomputing* chapter 3. (pp. 63–115). Cambridge, MA: MIT Press.

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*, 283–328.

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*, 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*, 46–59.

Page, M. P., & Norris, D. (1998). The primacy model: A new model of immediate serial recall. *Psychological Review*, *105*, 761–781.

Pascanu, R., & Jaeger, H. (2011). A neurodynamical model for working memory. *Neural Networks*, *24*, 199–207.

Ponzi, A. (2008). Dynamical model of salience gated working memory, action selection and reinforcement based on basal ganglia and dopamine feedback. *Neural Networks*, *21*, 322–330.

Redgrave, P., Prescott, T. J., & Gurney, K. N. (1999). The basal ganglia: A vertebrate solution to the selection problem? *Neuroscience*, *89*, 1009–23.

Reggia, J. A., Sylvester, J. C., Weems, S. A., & Bunting, M. F. (2009). A simple oscillatory short-term memory. In *Proc. AAAI Symposium on Biologically Inspired Cognitive Architectures II* (pp. 103–108).

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. *Proc. Nat'l Acad. Sci. USA*, *102*, 7338–43.

Roy, A. (2008). Connectionism, controllers, and a brain theory. *IEEE Transactions on Systems, Man and Cybernetics, Part A*, *38*, 1434–1441.

Sandberg, A., Tegnér, J., & Lansner, A. (2003). A working memory model based on fast Hebbian learning. *Network: Computation in Neural Systems*, *14*, 789–802.

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*, 3609–3615.

Schoofs, D., Preuß, D., & Wolf, O. T. (2008). Psychosocial stress induces working memory impairments in an n-back paradigm. *Psychoneuroendocrinology*, *33*, 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.

Sherman, S., & Guillery, R. (2006). *Exploring the Thalamus and Its Role in Cortical Function*. MIT Press.

Simen, P., & Polk, T. A. (2009). A symbolic/subsymbolic interface protocol for cognitive modeling. *Logic Journal of the IGPL*, *18*, 705–761.

Simen, P., Vugt, M. V., Balci, F., Freestone, D., & Polk, T. A. (2010). Toward an analog neural substrate for production systems. In *Proc. of the 10th Int'l Conf. on Cognitive Modeling* (pp. 223–228).

Singer, W. (2011). Dynamic formation of functional networks by synchronization. *Neuron*, *69*, 191–193.

Sporns, O. (2011). *Networks of the Brain*. MIT Press.

Stemme, A., Deco, G., & Busch, A. (2007). The neuronal dynamics underlying cognitive flexibility in set shifting tasks. *Journal of computational neuroscience*,

23, 313–31.

Stewart, T. C., Choo, F.-X., & Eliasmith, C. (2010). Symbolic reasoning in spiking neurons: A model of the cortex/basal ganglia/thalamus loop. In *Proc. 32nd Ann. Conf. Cognitive Science Society* (pp. 1100–1105).

Stewart, T. C., & Eliasmith, C. (2011). Neural cognitive modelling: A biologically constrained spiking neuron model of the tower of hanoi task. In *Proc. 33rd Ann. Conf. Cognitive Science Society* (pp. 565–661).

Stewart, T. C., Tang, Y., & Eliasmith, C. (2011). A biologically realistic cleanup memory: Autoassociation in spiking neurons. *Cognitive Systems Research*, *12*, 84–92.

Storkey, A. (1997). Increasing the capacity of a Hopfield network without sacrificing functionality. *Proc. ICANN*, (pp. 451–456).

Storkey, A., & Valabregue, R. (1999). The basins of attraction of a new Hopfield learning rule. *Neural Networks*, *12*, 869–876.

Sun, R. (2006). The CLARION cognitive architecture: Extending cognitive modeling to social simulation. In R. Sun (Ed.), *Cognition and Multi-Agent Interaction* (pp. 79–99). New York: Cambridge University Press.

Sylvester, J. C., Reggia, J. A., & Weems, S. A. (2011). Cognitive control as a gated cortical net. In *Proc. 2nd Int'l Conf. Biologically Inspired Cognitive Architectures* (pp. 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 D. D. Salvucci, & G. Gunzelmann (Eds.), *Proc. 10th Int'l Conf. Cognitive Modeling* (pp. 241–246).

Tanji, J., Shima, K., & Mushiake, H. (2007). Concept-based behavioral planning and the lateral prefrontal cortex. *Trends in Cognitive Sciences*, *11*, 528–534.

Tsuda, I. (2001). Toward an interpretation of dynamic neural activity in terms of chaotic dynamical systems. *The Behavioral and Brain Sciences*, *24*, 793–810.

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.

Watter, S., Geffen, G. M., & Geffen, L. B. (2001). The n-back as a dual-task: P300 morphology under divided attention. *Psychophysiology*, *38*, 998–1003.

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*, 161–171.

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, 741–761.

Womelsdorf, T., & Fries, P. (2009). Selective attention through selective neuronal synchronization. In M. Gazzaniga (Ed.), *The Cognitive Neurosciences* (pp. 289–302). MIT Press.