Neural Networks 79 (2016) 37–52

Contents lists available at ScienceDirect

Neural Networks

journal homepage: www.elsevier.com/locate/neunet

Engineering neural systems for high-level problem solving

Jared Sylvester, James Reggia *

Department of Computer Science, University of Maryland, A.V. Williams Building, College Park, MD 20742, United States

ARTICLE INFO

Article history:
Received 16 December 2015
Received in revised form 11 March 2016
Accepted 17 March 2016
Available online 31 March 2016

Keywords:
Neuroengineering
Attractor neural networks
Gated neural networks
Top-down vs. bottom-up AI
Neural network problem solving

ABSTRACT

There is a long-standing, sometimes contentious debate in AI concerning the relative merits of a symbolic, top-down approach vs. a neural, bottom-up approach to engineering intelligent machine behaviors. While neurocomputational methods excel at lower-level cognitive tasks (incremental learning for pattern classification, low-level sensorimotor control, fault tolerance and processing of noisy data, etc.), they are largely non-competitive with top-down symbolic methods for tasks involving high-level cognitive problem solving (goal-directed reasoning, metacognition, planning, etc.). Here we take a step towards addressing this limitation by developing a purely neural framework named GALIS. Our goal in this work is to integrate top-down (non-symbolic) control of a neural network system with more traditional bottom-up neural computations. GALIS is based on attractor networks that can be "programmed" with temporal sequences of hand-crafted instructions that control problem solving by gating the activity retention of, communication between, and learning done by other neural networks. We demonstrate the effectiveness of this approach by showing that it can be applied successfully to solve sequential card matching problems, using both human performance and a top-down symbolic algorithm as experimental controls. Solving this kind of problem makes use of top-down attention control and the binding together of visual features in ways that are easy for symbolic AI systems but not for neural networks to achieve. Our model can not only be instructed on how to solve card matching problems successfully, but its performance also qualitatively (and sometimes quantitatively) matches the performance of both human subjects that we had perform the same task and the top-down symbolic algorithm that we used as an experimental control. We conclude that the core principles underlying the GALIS framework provide a promising approach to engineering purely neurocomputational systems for problem-solving tasks that in people require higher-level cognitive functions.

1. Introduction

Most artificial intelligence (AI) and cognitive modeling systems fall into one of two general groups: systems that take a symbolic, top-down approach, and those that adopt a neural, bottom-up approach. The divide between these two strategies is both long-standing and, at times, quite contentious. This conflict is regrettable because the two different strategies are in many ways complementary rather than competitive: each of the two approaches has its own relative strengths and weaknesses. For example, while neural systems excel at problems that involve pattern matching, incremental learning, low level control, fault tolerance, and/or processing noisy data, they are less adept at handling higher cognitive functions such as goal-directed reasoning, meta-cognition, and planning. Top-down symbolic methods are largely just the opposite. This complementarity has been recognized in the past (Reggia, Monner, & Sylvester, 2014) and leveraged effectively in a number of cognitive architectures (e.g., Sun & Naveh, 2004).

The current limited abilities of neural architectures to capture critical aspects of high-level cognition put them at a tremendous disadvantage relative to symbolic AI techniques when trying to engineer neurocomputational systems for high-level problem-solving tasks. Such problem solving by people depends on *cognitive control*, the process of managing other cognitive processes (Schneider & Chein, 2003). Examples of cognitive control include such *executive functions* as shifting attention, response selection, working memory maintenance, goal setting, and inhibition of irrelevant signals. Executive functions are primarily associated with prefrontal cortex in the primate brain, and substantial recent work has focused on localizing these functions to specific individual prefrontal regions (Burgess, Dumontheil, & Gilbert, 2007; Koechlin & Summerfield, 2007).

* Corresponding author.
E-mail addresses: jared@jsylvest.com (J. Sylvester), reggia@cs.umd.edu (J. Reggia).

http://dx.doi.org/10.1016/j.neunet.2016.03.006
0893-6080/

The limited ability of neurocomputational methods to support higher-level cognitive/executive functions is somewhat surprising in that the human brain handles such issues routinely, thereby establishing that neural computations have the capacity to do so. In the work that we describe here, while we take inspiration from human cognition and neuroscience, we are not trying to create accurate models of either. Instead our primary focus is on how to construct/engineer neural network systems for problem-solving tasks that are competitive with top-down symbolic AI problem-solving systems. Developing purely neurocomputational systems for high-level problem solving could ultimately provide several significant practical advantages. For example, when compared to traditional top-down AI, neural computation is fault tolerant, and it has the potential for great speed due to its inherent parallel processing (Haykin, 2009; Reggia et al., 2014). The latter is particularly true given the recent increasing availability of parallel computing hardware such as neural network chips and GPU clusters. Further, neurocomputational methods have the ability to learn and adapt, something that will be increasingly important in future AI systems of all kinds.

Studying neurocognitive architectures involving cognitive control is currently viewed as an important research direction (Roy, 2008), and the importance of developing neural computational methods for cognitive control is likely to substantially increase over the next decade as work on developing large-scale brain and neurocognitive models accelerates. By *large-scale models* we mean recent and ongoing research efforts to create neuroanatomically grounded simulations of all or major portions of human/mammalian brain structure and function, or at least major subsystems of the brain that span multiple cortical regions. These models vary from extremely large networks of biologically-realistic spiking neurons to those that are more abstract, based on a higher level of components such as cortical columns, or are focused on simultaneously supporting human cognitive functions (e.g., de Garis, Shuo, Goertzel, & Ruiting, 2010; Eliasmith et al., 2012; Townsend, Keedwell, & Galton, 2014; Weems & Reggia, 2006; Winder, Cortez, Reggia, & Tagamets, 2007). They are often inspired by the view that the brain is organized as a network of regions that are inter-connected via well recognized pathways. Specifically, the primate cerebral cortex is organized as a distributed network of interacting cortical regions, exhibiting both functional integration and functional segregation (Bressler & Menon, 2010; Sporns, 2011; van Essen, Anderson, & Felleman, 1992). *Large-scale region-and-pathway models* inspired by this viewpoint consist of components (modules) that are neural network simulations of individual brain regions, e.g., Wernicke's and Broca's areas and the arcuate fasciculus that connects them (Monner & Reggia, 2013; Weems & Reggia, 2006). Work on large-scale neurocognitive models is increasing in part due to recent major funding initiatives (Europe's Human Brain Project, US BRAIN Initiative, etc. (Abbott, 2013)).

A fundamental question arises in constructing large-scale neurocognitive architectures like these: Is there an identifiable minimal set of generic, region-level functions and interactions that can be used to construct neural architectures that provide cognitive control of human-like problem-solving behaviors? As one possible approach to answering this question, we have recently proposed the GALIS framework (Sylvester, Reggia, Weems, & Bunting, 2013), where GALIS is an acronym for "Gated Attractors Learning Instruction Sequences". The methods used in GALIS are intended to provide a general purpose framework within which models for specific higher-level problem solving tasks can be instantiated and trained using solely subsymbolic methods. While it takes inspiration from the human brain and cognition, GALIS is *not* intended to be a veridical model of either the brain or human reasoning. The central issue that GALIS addresses concerns how one can adopt and extend purely neurocomputational methods to engineer high-level cognitive control of the sort that can currently only be readily modeled using top-down symbolic approaches. GALIS assumes that one is interested in constructing a large-scale region-and-pathway model of some aspect of human-level cognition that is inspired by the organization of the cerebral cortex, and perhaps other subcortical brain regions. An implication of this assumption is that model brain regions must learn not only the facts about a specific instance of a task, but also the procedure or "instruction sequence" that is needed to perform that task in general. This focus on making problem solving dependent on patterns stored in the network's memory, rather than on the network's structure or "hardware", differs from many previous models of cognitive control, and is intended to make GALIS models more generalizable: Their behavior can be changed by adjusting which behavioral sequences are learned rather than by adjusting the structure of the model itself.

GALIS answers the fundamental question above by adopting two principles. First, GALIS assumes that each region in the cortical network can be conceptualized as an attractor neural network — a dynamical complex system whose activity is driven towards certain preferred states. Attractor networks have been used previously in cognitive control models (Farrell & Lewandowsky, 2002; Hoshino, Usuba, Kashimori, & Kambara, 1997; Jones & Polk, 2002), but usually operate only with fixed-point attractors. In contrast, GALIS' attractors are designed to enable switching between attractor states in ordered sequences. This is critical if procedural information of the sort readily handled by top-down symbolic AI methods is to be accommodated in memory: procedures by their very nature must have temporal extents and their component steps must be performed in a specific order (Ismail & Shapiro, 2000). While multiple techniques have been used to add similar dynamism to attractor nets (Brown, Preece, & Hulme, 2000; Horn & Opher, 1996; Winder, Reggia, Weems, & Bunting, 2009), GALIS uses an approach to learning of temporally asymmetric connection weights that we recently developed (Sylvester, Reggia, & Weems, 2011; Sylvester, Reggia, Weems, & Bunting, 2010a).

Second, GALIS assumes that each cortical region cannot 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 one cortical region can modulate the functions of other regions, or open/close the flow of information between other regions. The inspiration for adopting gating as a central aspect of neural problem-solving systems comes from past neuroscience research and neurobiologically-realistic computational models suggesting that gating is an important aspect of brain dynamics (Frank, Loughry, & O'Reilly, 2001; O'Reilly & Frank, 2006; Sherman & Guillery, 2006; Singer, 2011; Womelsdorf & Fries, 2009). While there have been numerous theories posited for the biological structures that could directly or indirectly underly such cortical gating, GALIS is agnostic about the particular physiological implementation. Rather, we take the existence of some such mechanism as given and implement gating as direct interactions between model cortical regions and their connecting pathways.

In summary, the basic hypothesis being explored via GALIS is that large-scale region-and-pathway models based on (1) representing procedures/programs as temporal sequences of attractor states, and (2) allowing model regions to gate the behavior of other model regions, provide a sufficient purely-neurocomputational framework for engineering autonomous problem-solving systems that can be competitive with the more traditional top-down symbolic problem-solving systems that are used in AI. While our previous work with GALIS was encouraging in showing that it could successfully support models for simple working memory applications such as the *n*-Back task used in psychological testing (Sylvester et al., 2011, 2013), such applications were too simple to seriously support this hypothesis; they had no need to generate

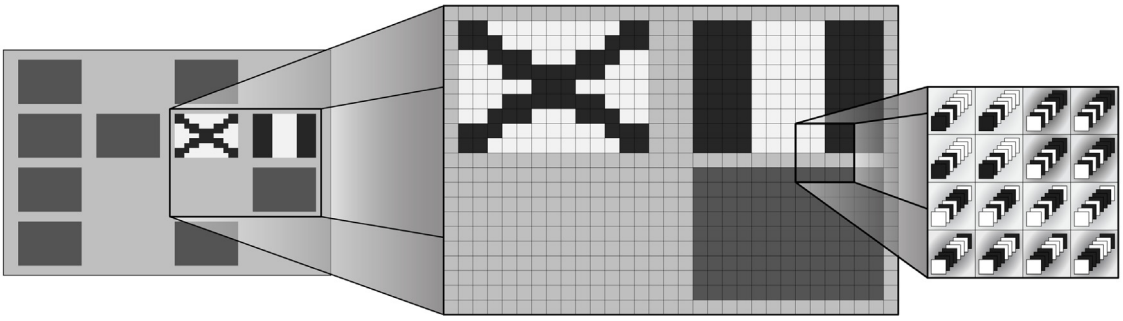


Fig. 1. A depiction of the visual environment for the card matching task during play. On the left is a view of the full environment. Here there are 10 cards still present, two of which are face-up showing an $\times$ and a vertically-striped pattern. The middle part of the illustration shows an expanded view of these two cards as well as one of the face-down cards. On the right a 4×4 section of the board has been enlarged to show how each of the four "colors" used in the visual field is encoded as an 8 bit binary string. In this example, the white areas on the faces of cards (upper left of detailed area) are encoded as 11010000 and the black areas on the faces of cards are encoded as 01010111.

outputs that influence the external problem state, they did not use spatial information, etc.

In this context, here we extend the GALIS framework in several ways beyond our previous work that only required determining when to learn or remove patterns from working memory in *n*-Back problems. Specifically, we construct a GALIS system for problem solving during much more challenging card matching tasks in which an agent uncovers pairs of face-down cards, trying to select pairs that have matching patterns on their faces. The extended system differs from our previous work in that now, for the first time, it incorporates interacting "what" and "where" visual pathways capable of associating seen objects with their locations, it retains in working memory the location and identity of previously-observed entities (uncovered cards), it uses top-down attention mechanisms to focus on specific portions of the visual environment, and it selects appropriate output actions to take to perform the card matching task correctly. Most importantly, the resulting GALIS system now effectively acts as an autonomous problem-solving system that interprets spatial relationships and controls output actions in an ongoing sequence of steps that sequentially alter the external problem state. Computational experiments are used to show that our GALIS card matching system not only performs the task successfully, but also that it does so in ways that are reminiscent of what we observed when we asked human subjects to perform this same task and when a top-down symbolic program is used to solve the same kind of problems.

2. Methods

Here we apply the GALIS approach for the first time to a complex problem-solving task known by many names including "Pairs", "Pelmanism", and "Concentration", but which we refer to as the *card matching problem*. This task involves first randomly placing several pairs of cards face-down on a tabletop. The player turns over two cards each round, one at a time, with the goal being to uncover matching pairs of cards so that they can be removed from the table. The problem is solved when all cards have been removed.

The requirements of the card matching task substantially expand on previous work with GALIS by placing stimuli patterns in a spatial environment. That is, the model is not just attempting to remember and work with a set of abstract stimuli in the æther, but must successfully bind together what was seen with where it was seen in the environment. The binding together of multiple features (Feldman, 2013), something readily handled by top-down symbolic AI methods, is an ongoing challenge for neural models. This task also requires that a system makes judgements about the contents of its own memory. In other words, it is not enough to just store a series of stimuli, but the model must be able to make strategic decisions based on introspection of its own memory of what cards have been seen previously and their locations, and then to take appropriate actions based on this information.

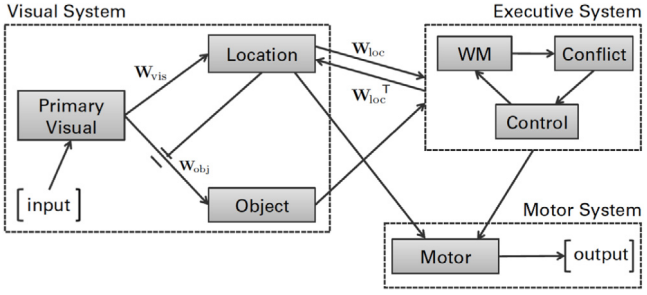


Fig. 2. Overall architecture of the GALIS model for the card matching task. Its seven modules form three functional systems, the visual, executive and motor systems. Each system is indicated by an enclosing dashed-line box. See Figs. 3 and 4 for more detailed views of the visual and executive systems, respectively.

In this work we are not concerned with the details of low-level image processing or low-level movement control, tasks at which existing neurocomputational methods are quite effective. Accordingly, our low-level sensory processing is intentionally greatly simplified. For example, in our implementation of card matching described below each card is only 9 by 13 pixels in size. The backs are a uniform dark grey, and the fronts are monochromatic patterns such as horizontal stripes, crosses, or diagonals. Depending on the experiment either four, six or eight pairs of matching cards (i.e., eight, twelve or sixteen cards in all) are arrayed on the table top (see Fig. 1), initially all face down. The images on the fronts of cards consist of monochromatic, low-resolution simplifications of national flags. For instance the $\times$-shaped card in Fig. 1 is derived from the flag of Scotland, while the striped card could alternately represent the flag of Italy, Ireland or France.

The GALIS model for card matching is composed of seven modules as illustrated in Fig. 2. These are the Visual, Location, Object, Motor, Working Memory, Conflict, and Controller modules. Their functions are explained in the remainder of this section.

2.1. Visual system

The Visual System in our model consists of the Primary Visual, Location, and Object modules, as shown in Fig. 3. The Primary Visual module provides input to the model. Its visual field consists of a 55×67 grid of grayscale "pixels". This size, and that of several other modules, is somewhat arbitrary: It was chosen so that the maximum number of cards used in the computational experiments described later could comfortably fit when displayed. Each primary visual node may take one of four values: light grey, representing the surface of the tabletop; dark grey, representing the back of a

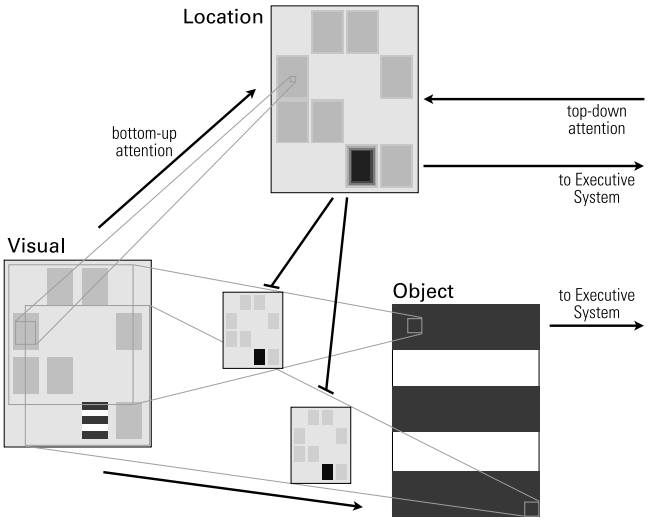


Fig. 3. The model's visual system. Here there are 8 cards depicted in the environment. All are face-down except for one in the bottom row, which is striped. The region of the Location module corresponding to this card is the most active. A Location node along the left edge is also highlighted, along with its topographically corresponding receptive field in the Visual module. The state of the Object module reflects that it is attending to the sole face-up card. Two of its nodes and their receptive fields are shown. The smaller rectangles between the Visual and Object modules are depictions of the incoming weights to these two nodes, as set by the Location module. One can view these as a cross-section of the connections, with just one of the many incoming links to each node being open/active.

card; white and black, which together make the patterns on the front of the cards. Each of these values is represented by a random 8-bit pattern, so there is a total of 29 480 nodes/neurons in the Primary Visual module.

2.1.1. Overview

The Visual module's output is sent to two different regions, the Location and Object modules, for further processing (Fig. 3). This parallel visual pathway organization is inspired by the ventral "what" and dorsal "where" visual pathways of the mammalian brain (Baizer, Ungerleider, & Desimone, 1991; Ungerleider & Haxby, 1994) and how these pathways provide integrated information to prefrontal cortical regions (via the temporal and parietal cortical regions). Binding the general location information provided by the dorsal pathway with the appropriate detailed object-specific information provided by the ventral pathway is a significant challenge (Reynolds & Desimone, 1999). In our model, the Location and Object modules can be seen as simplified analogs to the parietal and temporal cortices, respectively (Baizer et al., 1991). The former is responsible for broad but low-resolution vision—identifying that there is an object at a particular location, but not the particular features of that object – while the latter provides a detailed but narrow view – thus being able to discern details of the object but remaining ignorant of its location. The two visual pathways influence each other, with the Location module helping to guide the attention of the Object module, and the Object module providing detailed information about the visual field that the Location module lacks. This inter-pathway influence fits naturally with GALIS' use of gating/higher-order network connections. It is similar in spirit to past more-biologically realistic models of interacting dorsal and ventral pathways in the brain that have been studied in isolation (i.e., not in the context of problem solving as we do here) in order to better understand the neurobiology of visual processing (Heinke & Backhaus, 2011; van der Velde & de Kamps, 2001).

The Location module is the same size as the Visual module: 55 × 67 pixels. Rather than using 8 binary nodes to represent each pixel, each pixel is congruent with a single node with values in [0, 1]. This value roughly represents the salience the model gives to that location in the visual field. Nodes' activations are determined by the logistic sigmoid of the sum of both bottom-up input from the Visual Module and top-down attentional input from the Executive System. For the former of these inputs, each node is connected topographically, with a receptive field in the Visual module of a 5 × 5 square. The second input is from the Executive System. This input is gated so that when it is closed the attention of the model emerges from the interaction of the Visual and Location modules. When it is open, it serves to control visual attention from the top-down. This is of particular use when there are two cards face-up. Using only the bottom-up attentional mechanism both will be equally salient. The addition of the top-down element allows the Location module's focus to be directed to the card chosen by the Executive System.

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

The second output from the Location module is used, as mentioned previously, to control the receptive field of the Object module (see Figs. 2 and 3). The Object module (9 × 13 pixels, each of which uses 8 binary nodes to encode "color" values, for a total of 936 nodes; these sizes are somewhat arbitrary) can access the visual field in finer detail, but at the cost of a limited scope. The Location module determines where the Object module should focus its limited field-of-view. We view this as an example of gating. The output of the Location module is able to open and close activity flowing from the Visual to the Object module. Thus, $\mathbf{W}_{\text{obj}}$ acts like a mask, only allowing the portion of an Object node's receptive field which corresponds topographically to a card of interest to pass through. Activity in some portion of the Location module (which manifests as a rectangularly-shaped block of activation) moves the focus of the Object module to attend to the area in the visual field corresponding to this activity. So, for example, if there is high activity in the upper left of the Location module then the connections between the upper left of the visual field and the Object module are opened and the rest is closed. This gating can be viewed as an example of higher-order nodes (Lipson & Siegelmann, 2000). It is implemented via multiplicatively modulated weights (Akam & Kullmann, 2014), in which one network's output (from the Location module) is used to adjust the weights of another network (the Object module). The Object module is thus guided to focus on a particular region of the input plane by the Location module using a combination of the bottom-up information from the Primary Visual module and top-down information from the Executive System. Its output proceeds upstream to the Working Memory region with one-to-one connections, so that the model can form a memory of the visual appearance of cards it has seen.

2.1.2. Location Pathway

The weights on the bottom-up connections ($\mathbf{W}_{\text{vis}}$) from the Visual to Location modules are trained using one step of Hebbian learning per input pattern. We use extensive weight-sharing, allowing nodes to have identical incoming weights but different receptive fields. This makes training efficient. The training patterns are a selection of possible 5 × 5 patterns which appear on cards (12.5% of the total possible patterns are used for training). Through this process nodes learn to turn on when they detect patterns from the faces of cards, and remain off when their input field is the table top. This produces a rectangular surge of activity in the Location

module which corresponds topographically to that of face-up cards in the Visual module. (In Fig. 3, this is the horizontally-striped card in the center-right of the bottom row.)

The weights on the top-down connections to the Location module from the Executive system are merely the transpose of the weights on the counter-flowing, bottom-up connections discussed immediately below. The final state of a node is just the sum of both the bottom-up and top-down influences, weighted by the appropriate gating factor.

$$l_i = \sigma\left(\mathbf{W}_{\text{vis}}\, R(i) + g_{\text{WM,loc}}\, \mathbf{W}_{\text{loc}}{}^{\mathsf{T}}(:, i)\, \vec{a}\right). \tag{1}$$

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

The output from the Location module to the Executive system is thus the average of each node's weighted activity. As a result, overlapping areas of activity produce similar outputs, despite the randomness of each individual node's representation. This system also has the desirable by-product of reducing the dimensionality of the spatial encoding from that needed by the Location module (3685) to that used by the Working Memory. The overall effect is similar to that of Random Matrix Transformations (Achlioptas, 2003; Johnson & Lindenstrauss, 1984; Rahimi & Recht, 2008). Finally, this approach has the added advantage of being easily invertible: $\mathbf{W}_{\text{loc}}$ is used to translate between the encoding used by the Location Module to that used by the Working Memory, providing bottom-up visual attention; $\mathbf{W}_{\text{loc}}{}^{\mathsf{T}}$ is used for the reverse translation, allowing for top-down attention control.

2.1.3. Object pathway

Every node i in the Object module has a receptive field of 47×55 nodes in the Visual module. At any one time, each Object module node should only be accepting input from one of those Visual module nodes. Furthermore, each Object node should be accepting input from the node in the same location in its receptive field—i.e., if one node is attending to the middle of the top row of its receptive field, so should the others be. Each area of activity in the Location module therefore translates into a single active point in a 47×55 grid (see Fig. 3). These pairs of active regions in the Location module with their correlating points of focus in the visual field are used as training patterns for the hetero-associative Hebbian learning used to form the weights $\mathbf{W}_{\text{obj}}$ controlling the Object module's focus.

The activation o_i of Object module node i can be formalized as

$$o_i = \sigma\left(\sum_{j \in N(i)}\left(x_j \sum_{k \in \text{loc}} \mathbf{W}_{\text{obj}jk}\, l_k\right)\right) \tag{2}$$

where σ is again the logistic sigmoid function, $N(i)$ is the receptive neighborhood of node i, x_j is the state of a node in the Visual module, *loc* is the set of nodes in the Location module, and l_k is the state of one of the nodes in the Location module.

2.2. Executive system

The Executive System consists of three modules (see Fig. 2): Working Memory, a Controller, and a Conflict module. It is inspired by functionalities generally associated with prefrontal cortical regions of the brain. The Executive System's structure is shown in more detail in Fig. 4.

2.2.1. Working memory

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

Training of Working Memory is accomplished with one step of Hebbian learning per stored pattern, which occurs whenever the Working Memory training gate is open. This establishes the pattern to be learned as an attractor in the Working Memory's state space. This pattern can then be recovered when the network is in a state sufficiently close to it: either a noisy or corrupted version of the original pattern, or – more importantly for our purposes – when a part of the pattern in missing. For example, if the ×-shaped "Scotland" card depicted in Fig. 1 is observed in the top-most row and left-most column of cards in the environment, then the working memory would learn a string corresponding to the object–location tuple (×; top left corner). This string becomes stored in working memory, or in other words, it becomes an attractor state of the Working Memory network via one step of Hebbian learning. This allows the full pattern to be recovered from either portion, and explains how our model binds information together without using mechanisms such as oscillatory synchronization. That is, by subsequently setting the "what" nodes to "×" and allowing the memory to update, the "top left corner" portion can be recovered, and vice-versa, as illustrated in Fig. 5. It is in this sense that "×" and "top left corner" are bound together in the model's working memory. Working memory has 1960 nodes (936 what, 1024 where).

Every time the Working Memory is trained with an object–location pair, its prior weights undergo weight decay by an amount δ_{N}. This reduces the interference between patterns in memory, and allows for older memories to be supplanted by more recent ones. Adjusting the amount of decay can be used to affect the length of sequences that a memory like this stores: lower levels of decay allow longer series at the cost of more interference, while higher levels of decay are better suited for shorter series (Winder et al., 2009). More specifically, the weight matrix, W_{WM}, is trained with one-step of Hebbian learning per pattern $\vec{a}$ stored with the addition of a weight decay term so that older memories are supplanted by more recent ones:

$$w_{ij}(t) = (1 - \delta_{\text{N}})\, w_{ij}(t-1) + \frac{1}{N}\, a_i(t)\, a_j(t)\, (1 - \delta_{ij}) \tag{3}$$

where $a_i(t)$ is the activation level of the ith node and N is the number of nodes in the working memory module. Here δ_{N} is the decay rate ($0 \leq \delta_{\text{N}} < 1$) and δ_{ij} is Kronecker's delta, the latter ensuring that weights on self-connections stay fixed at zero.

The model's Working Memory also includes a "register" layer (see Fig. 4) to allow the comparison of one pattern to another. These allow operations on multiple operands. The register layer is the same size as the main WM layer, and is linked to it by one-to-one, gated connections allowing patterns to be read into and out of this extra buffer. This layer was not present in the earlier n-Back model described in Sylvester et al. (2013) as the task addressed there did

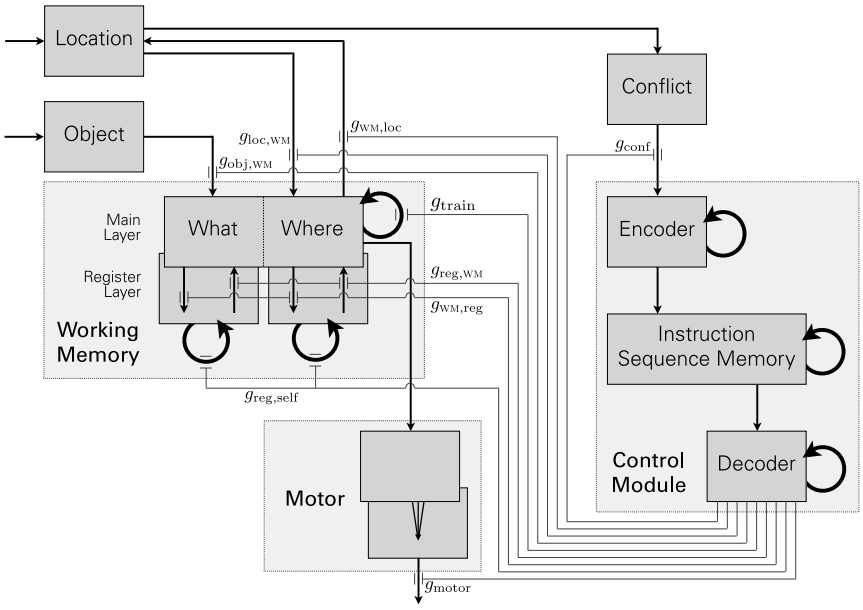


Fig. 4. Architecture of the model's executive system. Thick arrows represent full internal connections. Thinner gray lines represent gating outputs from the Controller. Note that the size of layers is not to scale.

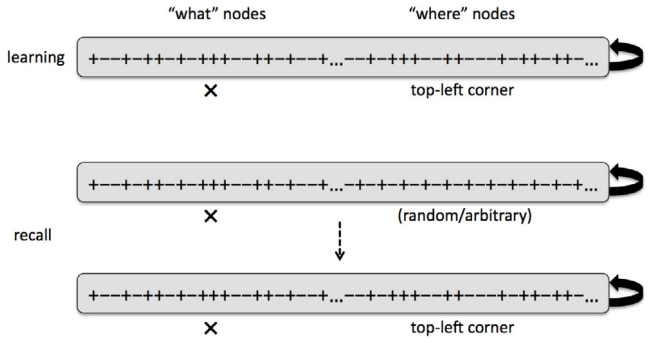


Fig. 5. A schematic illustration of how the binding of a flag's pattern (✕ in this case) and its location (top left corner) occurs in working memory. *Top:* To learn this association, the activity patterns representing ✕ and top left corner are input to the working memory. These patterns are composed of ±1 node values (here, a "+" indicates a +1 activation level and a "−" indicates a −1 activation level for a node). The working memory elements are fully connected to one another, as suggested by the curved solid arrow on the right, with weights W_{WM} on their connections. In other words, while we conceptually interpret the working memory as being composed of "what" and "where" nodes, all of these nodes are connected to one another. One step of Hebb-like learning as described in the text is used to make this composite activity pattern an attractor state of the network, thereby binding together the ✕ and top left corner information. Multiple what–where pairs of previously seen cards are stored like this simultaneously in working memory. *Bottom:* Once a what–where pair has been stored in working memory, it can be recalled by initializing the working memory with just one part of the pair. For example, here the "what" activity pattern ✕ is input to working memory along with an arbitrary/random activity pattern for its location, and the network's activity is allowed to change (vertical dashed arrow) until, biased by the initial ✕ activation sub-pattern, it reaches an attractor state representing the originally stored object–location tuple (✕; top left corner).

not require comparing multiple patterns stored within working memory.

Considered in more detail, the updated activity states s_i of nodes in the primary WM layer are the result of a sum, weighted by the appropriate synaptic strengths w_{ij} and gate values g, of the current states s_j, the state of the corresponding register node r_i, and the output of the visual system o_i. For nodes which encode "what" information, this latter value is simply based on the state of the Object Layer. For nodes encoding "where" information, it is the state of the Location Layer, as weighted by the $\mathbf{W}_{\text{loc}}$ weight matrix. This can be formalized as

$$s_i = \text{sgn}\left(\sum_{j\in\text{WM}} \mathbf{W}_{\text{WM}ij} s_j + g_{\text{reg,WM}} r_i + g_{\text{obj,WM}} o_i\right) \tag{4}$$

for "what" nodes and

$$s_i = \text{sgn}\left(\sum_{j\in\text{WM}} \mathbf{W}_{\text{WM}ij} s_j + g_{\text{reg,WM}} r_i + g_{\text{loc,WM}} \sum_{k\in\text{loc}} \mathbf{W}_{\text{loc}ik} l_k\right) \tag{5}$$

for "where" nodes. Here o_i is the activity of the ith node of the Object module, l_k is the activity of the kth node of the Location module, $g_{\text{reg,WM}}$is the gate on the register-to-WM connection, $g_{\text{obj,WM}}$ is the gate on the Object-to-WM connection, and $g_{\text{loc,WM}}$ is the gate on the Location-to-WM connection.

Nodes in the registers update their activity r_i according to the simple rule:

$$r_i = \text{sgn}(g_{\text{reg,self}}\, r_i + g_{\text{WM,reg}}\, s_i) \tag{6}$$

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

2.2.2. Controller overview

The Controller, which we synonymously refer to as the Control Module, is at the core of the Executive System (see Fig. 4). It is trained to direct the operation of the rest of the model by opening and closing nine gates that govern the flow of activity, thereby allowing the entire system to function autonomously after training. The structure and function of the Controller are unchanged from the previous GALIS instantiation of Sylvester et al. (2013), which was used to model performance of the n-Back task

from cognitive psychology (Owen, McMillan, Laird, & Bullmore, 2005). The identical Controller architecture, when trained initially on a different set of instruction patterns, was used to perform this very different task—i.e., the controller provides a general purpose mechanism for cognitive control.

The nucleus of the Controller is a discrete attractor network called the *Instruction Sequence Memory* (ISM). This is a memory unit whose purpose is to store the instructions for problem solving, in this case for card matching problems. A GALIS model's behavior is determined in large part by multiple sequences of patterns that have been stored simultaneously in its ISM, which shift the control of the network's behavior away from its architectural construction and towards its informational content. Informally, the ISM allows one to store (via one step of learning per instruction being stored) a procedure or "program" for solving problems such as the card matching task.

Input to the ISM comes from a subcomponent called the Encoder, which is a hetero-associative neural network that is responsible for converting the Controller's input into a selection of which stored instruction sequence the ISM will process. The ISM's output is sent to the Decoder, which provides a hetero-associative network that translates between the patterns stored in the ISM and the actual values which are sent to each gate. These two hetero-associative sub-components serve dual purposes. The first is noise reduction. This is possible because each specializes in either storing inputs or outputs, while the ISM is required to represent both inputs and outputs as well as the links between them. The second purpose is to decouple the size of inputs from outputs. For example, the Controller has nine outputs: one for each gate. However, more than nine nodes are needed to store a distributed representation of the six actions used for the card matching task in Hopfield-type attractor networks (Storkey, 1997). The Encoder and Decoder make it possible to translate between representations of differing sizes.

The core of the Controller, the ISM, is trained prior to task execution on the necessary sequences of steps the model must take to perform the card matching task. Each of the eight rows in Table 1 (i.e., the eight actions in the right column) is represented by a binary string, and each of these strings is stored as an attractor state of the ISM using one step of Hebbian learning per instruction. These eight actions are linked together into three different sequences using temporally asymmetric weights (Sylvester et al., 2011, 2010a), as explained in more detail below. Each of these sequences corresponds to the appropriate response to there being either zero, one or two cards face-up. Depending on the number of cards which are face-up – as judged by the Conflict Module – the Controller executes one of the three action sequences detailed in Table 1.

The central idea of the ISM's instruction sequence execution process can be illustrated schematically as state transitions over a changing energy landscape, as illustrated in Fig. 6. In Fig. 6(a), the ISM's initial state (open circle) quickly transitions to an attractor state (filled circle) at an energy minimum that is determined by the network's symmetric weights. This latter fixed-point attractor state corresponds to execution of the first learned instruction of the selected instruction sequence. While remaining in this attractor state, node thresholds gradually change (increasing for nodes with $+1$ activation level, decreasing for those with -1 activation level), effectively modifying the energy landscape so that the ISM's state no longer lies at an energy minimum (open circle in Fig. 6(b)). Guided by the learned asymmetric weights on connections, the ISM's state transitions to another energy minimum that represents the next stored instruction (Fig. 6(b)). This process continues repeatedly as the ISM executes its "stored program" (Fig. 6(c), where the filled circle corresponds to the activity pattern of the third instruction that is executed).

The ISM's outputs are the states of the nine gates distributed throughout the rest of the model (see Fig. 4). They are:

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

This set of gates is manipulated in order to act out the set of six different actions the Controller takes in execution of card matching. These six actions are combined in different ways to create the three sequences in Table 1.

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

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

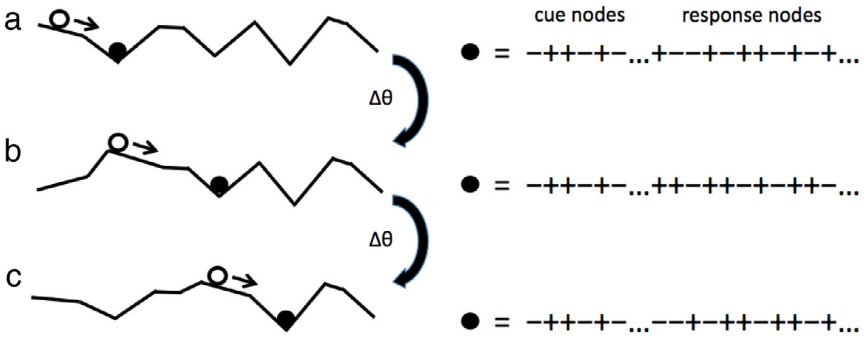


Fig. 6. Schematic illustration of ISM state transitions during execution of a sequence of instructions. On the left, the horizontal axis in each case represents the ISM's activity state, while the vertical axis represents the corresponding energy level. The irregular horizontal lines thus correspond to the energy landscape at three different times associated with different activity states of the ISM. Local minima of an energy landscape correspond to attractor states of the ISM that represent stored instructions. The circles represent the ISM's current state (open circle = transient non-attractor state; filled circle = fixed-point attractor state for the pictured energy landscape). On the right, example encodings of individual instructions stored in an instruction sequence in the ISM. Each instruction, which is represented as a randomly chosen activity pattern of ± 1 node values, conceptually consists of cue node values followed by response node values. A "+" indicates a $+1$ activation level and a "−" indicates a -1 activation level for a node. Cue node patterns are identical for instructions in a single sequence as shown here. Response nodes are unique to each instruction, and determine the gates that are opened/closed by the instruction. **a**. The ISM's initial state (open circle) quickly transitions to an attractor state (filled circle) at a local energy minimum determined by the ISM's learned symmetric weights W_{ISM}. This state corresponds to a stored instruction, which is pictured to the right. **b**. While the ISM remains in the attractor state of (a), threshold values slowly change (thick arrow labeled $\Delta\theta$). The changing thresholds effectively modify the energy landscape (compare energy landscapes in (a) and (b)), so that the initial instruction in (a) is no longer a stable equilibrium point of the ISM in (b). The ISM's activity pattern (open circle), guided by the ISM's learned asymmetric weights V_{ISM}, transitions to a new fixed point (filled circle) that corresponds to the second instruction in the sequence, which is pictured on the right. **c**. This process repeats itself, here showing the transition to the third instruction, until the instruction sequence has ended.

Table 1
Learned response sequences stored in the control module's ISM.

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

the external environment it appears to be a randomly guessed location. This has the desired affect of making the model guess a location to explore. In either case, the "where" sub-pattern which results from this update is passed to the Motor module for output and back to the Location module. In both the zero and one card visible situations the Working Memory's attractor state is being used as both a store of memory and a decision-maker, in a way similar to the network described in Machens, Romo, and Brody (2005). The dynamic network described there performs a sequence of operations described as 'perceive-hold-compare'. The WM of GALIS can be seen as performing a similar '*recall*-hold-compare' set of operations.

The model's actions when there are two cards face-up are very simple (as they are for a human player). If the two cards match then they are removed from the environment. If not, the only action required is to add the newly observed card to Working Memory. This is accomplished in the same way as with the beginning of the one-card-up case: open the gates allowing the Object and Location Modules to load their activity in the Working Memory and then update the Working Memory weights on this newly observed pattern.

To illustrate how the trained control module works during execution of a task, in Fig. 7 we show an example of how the model responds to a simplified environment of four cards, one of which is face-up. The key gates involved at each step, including those undergoing changes, are high-lighted by small, gray-shaded circles. The steps depicted here do not correspond one-to-one with time steps as they are simulated within the model: we have broken them down further to allow us to narrate the interactions more clearly. The example begins with one striped card face-up in the visual field (Fig. 7(a)). The Location module identifies this card as being the most salient place to focus on, and guides the Object module to attend to it. The existence of only a single location of interest also results in a low output from the Conflict module, which consequently directs the Control module to begin processing Sequence 2 in Table 1. Once Sequence 2 has begun (Fig. 7(b)), the Encoder Update gate is closed until the sequence is complete. Action 4 of Sequence 2 dictates that the gates connecting the Location and Object modules with the corresponding portion of Working Memory be opened. The resulting input causes the Working Memory layer to adopt the attractor patterns representing a horizontal striped card located in the lower left corner. Action 5

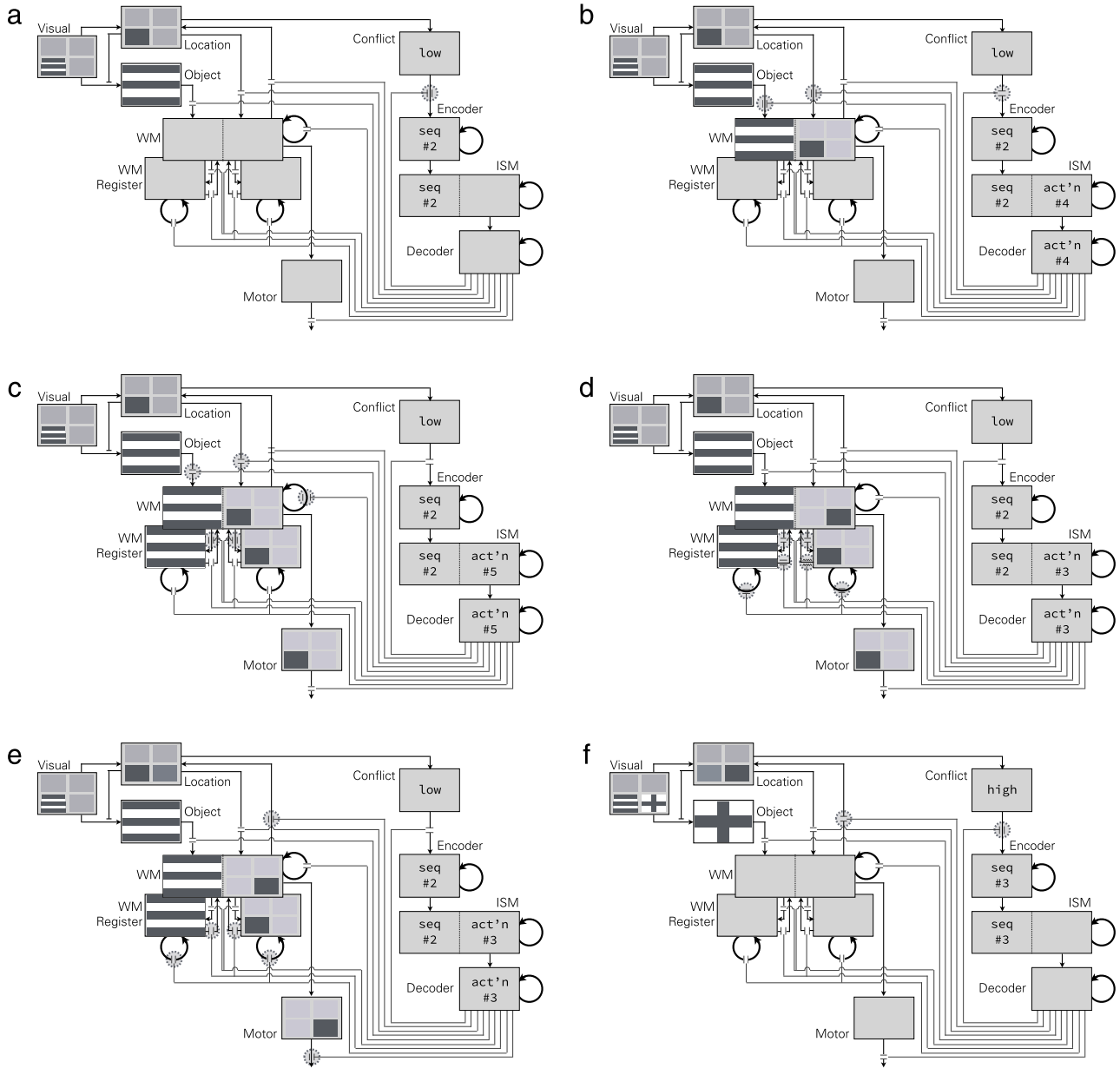


Fig. 7. Step-by-step operation of the model to process one card. (See text for details.)

instructs the Working Memory to train on the current pattern (Fig. 7(c)), adding the currently viewed card to its record. The current pattern is also loaded into the Register layer. The Object- and Location-to-Working Memory gates are then closed. The Register-to-Self gates are opened (Fig. 7(d)), allowing the Registers to maintain their current states. The WM-to-Register gates are closed, but the reverse Register-to-WM gates are opened. The Object portion uses excitatory activity to influence the main Working Memory layer to maintain the horizontal stripes pattern. The Location portion uses inhibitory activity to try and push the Working Memory layer out of the lower-left-corner attractor. Ideally when the WM is updated its state will shift to another attractor state which has been associated in memory with the same striped pattern. If no other location has been learned with the current object pattern then a location will be chosen randomly. In this case the Working Memory state updates so that the location portion represents the lower-right corner (Fig. 7(e)). The WM-to-Location gate opens, allowing top-down attention, and drawing the focus of the Location gate towards the lower-right card. The Motor Output gate is also opened, allowing the model to "gesture" to the lower-right card it would like revealed next. The card in the lower-right is revealed, showing a cross pattern (Fig. 7(f)). The combination of the bottom-up input from the Visual layer and the top-down direction from the Working Memory results in the Location module focusing on this new card. However, there is now a conflict between which of the two cards should be attended to, resulting in higher activation in the Conflict module, which causes Sequence 3 for the two-cards-face-up situation to be executed next.

2.2.3. *Instruction sequence memory (ISM)*

The ISM is a discrete autoassociative memory that uses temporally asymmetric learning in addition to standard Hebbian learning to process sequences (Sylvester, Reggia, Weems, & Bunting, 2010b). This allows it to store which actions make up the responses needed for a task, and also the order in which those actions must be carried out. Importantly, the ISM is capable of storing multiple action sequences at the same time. This is accomplished by way of a conceptual division of the ISM's nodes

into two sets, the "cue" and "response" nodes. 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 instructions 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, where an action can belong to more than one sequence.

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 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 associative memory. The weights on these connections are trained using one step of Hebbian learning per instruction 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. More specifically, the external input to ISM node i is defined by $e_i = a_{\text{enc}_i}$ for $i \in \{\text{CUE}\}$ and $e_i = g_{\text{conf}} \sum_{k \in \text{enc}} u_{ik} a_{\text{enc}_k}$ for $i \in \{\text{RESPONSE}\}$. Here 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_{conf}, described earlier, is also used for this additional encoder update gate (not pictured in Fig. 4).

Prior to performing any card matching tasks, the ISM is "programmed" by storing in it the appropriate instruction sequences that are needed. Each instruction that is to be stored is assigned to the ISM one time, and a single step of learning of the weights on the ISM's recurrent connections is done, updating its two weight matrices, $\mathbf{W}_{\text{ISM}}$ and $\mathbf{V}_{\text{ISM}}$. The former is trained using Hebbian learning and the latter using temporally asymmetric learning, as defined by the following rules:

$$w_{ij}(\tau) = (1 - k_{\text{ISM}})\, w_{ij}(\tau - 1) + \frac{1}{N}\, a_i(\tau)\, a_j(\tau)\, (1 - \delta_{ij}) \tag{7}$$

$$v_{ij}(\tau) = (1 - k_{\text{ISM}})\, v_{ij}(\tau - 1) + \frac{1}{N}\, a_i(\tau - 1)\, a_j(\tau) \tag{8}$$

where τ is the training step, $a_i(\tau)$ is the activation level of node i, and N is the number of ISM nodes. Here $\vec{a}$ is simply the concatenation of a cue and response pattern which make up one of the distinct actions listed in Table 1. The value k_{ISM} can be positive or negative. When negative, it acts as a gain rather than decay. Negative values of k_{ISM}, which serve to amplify earlier items, have surprisingly been found to be beneficial for the ISM; we use $k_{ISM} = -0.3$ here.

The dynamics of the ISM during subsequent task performance involves a two-part update process, first with the asymmetric and then with the symmetric weights, using

$$h_i(t) = \sum_j v_{ij}\, a_j(t - 1) - \theta_i(t) + e_i(t) \tag{9}$$

$$f_i(t) = \sum_j w_{ij}\, a_j(t) - \theta_i(t) + e_i(t) \tag{10}$$

for node i, where θ_i is a dynamic threshold that is used to keep the network from settling permanently into any one attractor basin.

More specifically, 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 $k_\theta = 0.02$ and $k_w = 0.0125$ are used. The input h_i computed as above is then used to update the state of each node according to

$$a_i(t) = \begin{cases} +1 & h_i(t-1) > 0 \\ a_i(t-1) & h_i(t-1) = 0 \\ -1 & h_i(t-1) < 0. \end{cases} \tag{11}$$

After updating both $\vec{a}$ and $\vec{\theta}$, the updating process begins again, this time using $f_i(t)$ computed as above to help the network settle further into the new attractor basin it was pushed towards by V_{ISM} in the previous stage. The asymmetric weights V_{ISM} suffice to get the network into the next attractor basin; the symmetric weights W_{ISM} impel it into the bottom of that basin, reducing the noisiness of the recall. This new input $f_i(t)$ is then used instead of $h_i(t)$ to update $\vec{a}$ according to Eq. (11) again (i.e., the conditionals are predicated on f_i, not h_i, this time), and $\vec{\theta}$ is updated once again.

2.2.4. Conflict module

The purpose of the Conflict Module (see Fig. 4) is to gauge the level of disagreement in the Location Module about where to focus attention. This gives an indication of how many cards are face-up in the visual field. When no cards are face-up, the Location module will have minimal activity, resulting in very low conflict. When one card is face-up, its location will be the single dominant source of activity, also resulting in low conflict. However, when two different card faces are visible, each location will vie for attention, causing internal disagreement about which to encode. The Conflict Module monitors for that disagreement, and reports it to the Controller. This behavior is inspired by the mismatch detection functions of the anterior cingulate cortex (Brown & Braver, 2005).

By informing the Control Module about how many cards are face-up, the Conflict Module allows the Controller to choose which of the three sequences described in the previous section it will execute. The connection from Conflict to Control Modules is gated; the signal to open the gate is only sent at the termination of each of the three instruction sequences. This results in the gate being open at the next time step, which allows the Controller to assess which sequence it should begin running. While this gate is closed, no input is received by the ISM, which allows it to continue running the current instruction sequence without interruption.

The inputs to the Conflict Module come from 512 randomly selected pairs of nodes in the Location Module, with each pair providing one bit of the eventual output. It would of course be possible to define the overall conflict to be a function of the entire Location module's state, but this global calculation over all possible pairs would be computationally expensive and was found to be unnecessary. The desired output differs depending on the distance between the two nodes in a pair. Nodes that are topologically close in the Location Module should have similar states; there is no inherent conflict in neighbors agreeing with each other. If nearby nodes have different states that is an indication of conflicting representations. In contrast, nodes that are far apart are not in conflict if they are both inactive, but are in conflict if they are both active as the latter represents the attempt to encode two disparate locations at once. The Conflict Module's final output is effectively the proportion of sampled pairs of nodes that are either nearby but in different states or far apart but both active.

Accordingly, the amount of conflict present in the Location module's activity is a function of the number of face-up cards being perceived. A single face-up card requires the Location module to

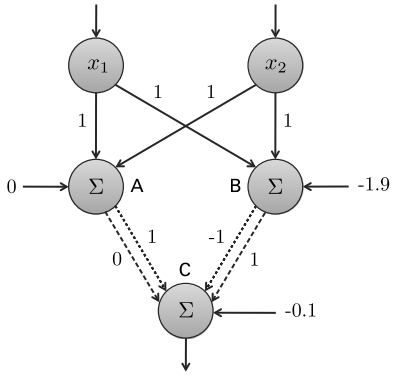


Fig. 8. Wiring of a pair of nodes in the Conflict module based on the activities x_1 and x_2 of two arbitrary nodes in the Location Module. Node A acts as a logical OR node and node B acts as a logical AND node. There is only one link from A to C and one from B to C, with the weights on these links depending on the distance between x_1 and x_2 in the Location module. The weights labeled on the dotted lines are used for Location nodes near each other, while the weights on the dashed lines are used for nodes which are far apart. Horizontal arrows designate bias values.

represent only a single region of space, while two face-up cards will result in two active representations. In the former case there will be only a single region of high activity in the Location layer, which results in a low-conflict 'consensus' about the spatial region to focus on. In the latter case there will be two regions of high activity in conflict with one another. It is the job of the Conflict Module to assess the amount of this conflict and report it to the ISM.

This is implemented as follows. The goal is to output $x_1 \oplus x_2$ if two selected Location module nodes 1 and 2 are within some topological distance of each other (here $\oplus$ indicates exclusive or), and to output $x_1 \wedge x_2$ if they are not. To accomplish this, each pair of nodes is connected by a network like that shown in Fig. 8, with the weights w_{CA} and w_{CB} set according to the distance between nodes 1 and 2. We use a Chebyshev distance equal to 7.5 to differentiate between "near" and "far". In other words, if $\|i, j\| < 7.5$ then $w_{CA} = 1$ and $w_{CB} = -1$, otherwise $w_{CA} = 0$ and $w_{CB} = 1$.

2.3. Motor system

The Motor System is intentionally very simple, consisting of only a single module (Fig. 2), because as with vision, detailed motor control is not being studied here. This Motor Module is roughly analogous to the premotor cortex. It consists of two layers (Fig. 4), both of which are the same size as the Visual and Location Layers (55×67). The first layer has inputs from the Working Memory which are encoded using the same methods as the top-down mechanism linking the Working Memory to the Location module (i.e., assigned $\mathbf{W}_{loc}{}^{T}$). The second layer is connected topographically to the first, with each node being linked to a 5×5 rectangle of nodes below it, each with a fixed and equal weight. This blurs the first layer's representation of the output space, allowing smoother, less volatile output.

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

2.4. Experimental methods

We experimentally assessed whether the GALIS framework could successfully solve card matching problems, and if so, how its solutions compared to two other methods for performing the task, where performance was measured as the number of rounds required to remove all cards. The two other approaches to solving card matching problems that we used as "gold standards" served as experimental controls for comparison purposes, as follows. First, in order to assess GALIS' performance similarities to and differences from people, we collected data from human subjects as they performed a web-based version of the card matching task that we developed.[1] The 34 participants played a total of nine times, three each with either 8, 12 or 16 cards on the board. This gave us 102 recorded trials for each of the three conditions. The ordering of trials was randomized for each subject to minimize biases due to ordering effects. The images used on the human subjects cards were randomly selected each trial from 10 pairs of national flags. To remove any potential influence of disparate hues and to better match the monochrome inputs of the neural model, all of the flags were composed of red, white and blue only.

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

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

The GALIS results presented below are averages from 200 simulation runs of the model. In each case the model's ISM was pre-trained on the necessary instructions, which were identical for all three task variants. With 8 and 12-card variants, the locations of the cards were randomly chosen from among the positions used for the 16-card case.

3. Results

The primary result of this work is that the GALIS system successfully solved every one of the hundreds of randomly-generated card matching tasks on which it was tested. The number of rounds it needed to complete the task, averaged over 200 runs for each number of cards n initially present, were 8.7, 13.0 and 21.2 for 8-, 12- and 16-card versions, respectively. This compares to mean human scores for our subjects of 7.9, 13.5 and 18.9. These results with our GALIS system were achieved with a single working memory decay rate $\delta_N = 0.125$, i.e., it was not necessary to adjust

[1] This can be accessed at http://www.jsylvest.com/cards/.

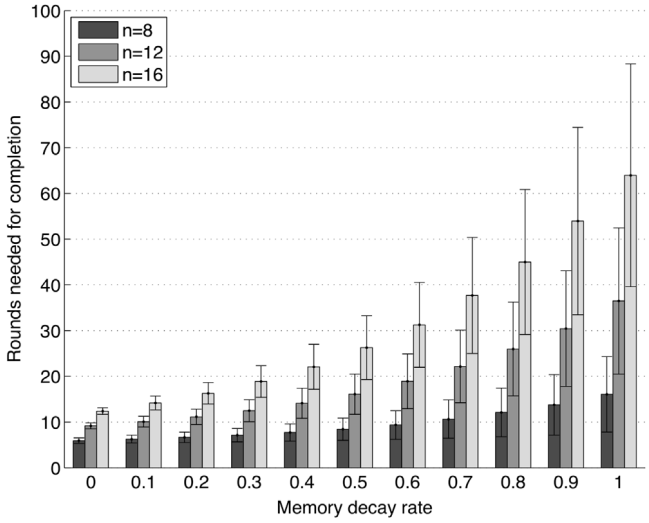


Fig. 9. Performance of the symbolic model on all three card matching task versions with varying decay rates. Note that as decay increases, both the expected performance (rounds needed) as well as deviation in performance increases. Based on 200 runs of the symbolic model for each value of the number of cards n. Error bars represent standard deviations.

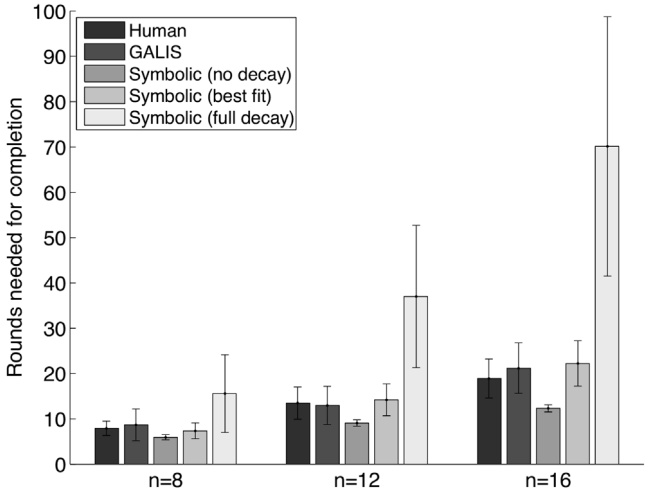


Fig. 10. Mean and standard deviation for human subjects, as well as symbolic and GALIS models on the card matching task for the $n = 8$, $n = 12$ and $n = 16$ conditions. For all three n conditions the decay rate of the GALIS model was 0.125. Results from the symbolic model are shown when it experiences no decay to its memory ("perfect play"), a decay rate of 0.375 (best fit to human performance), and complete memory decay (random play). By adjusting the decay rates of both models it was possible to produce better fits to the human data, but the values used here provided the best fit across all three n conditions without varying the decay rate. The difference between the human results and those of the GALIS model and the best-fitting symbolic model are significant only in the $n = 16$ condition.

the parameters, structure or instructions of the model in any way to perform across all the three conditions.

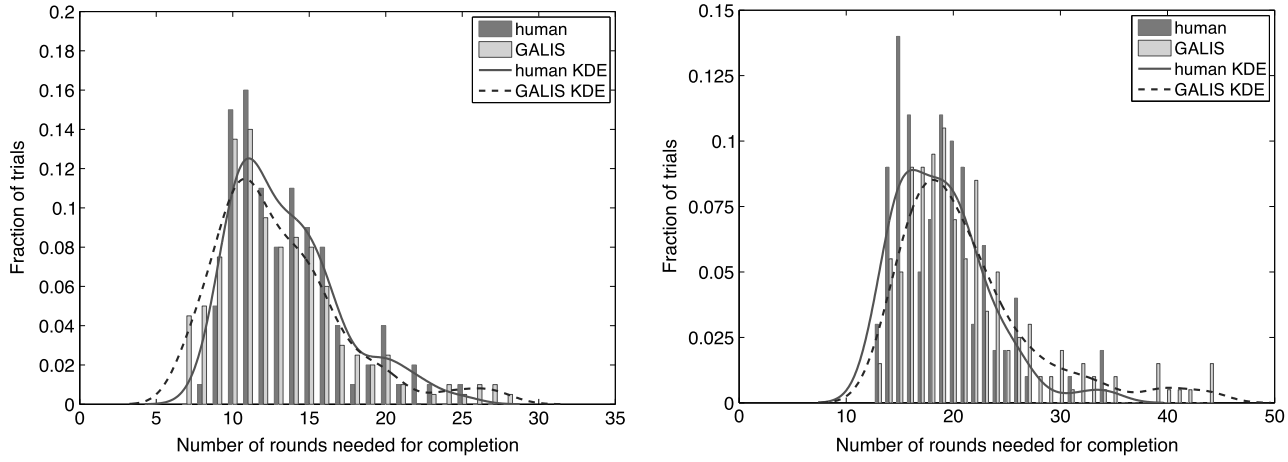
Prior to comparing our GALIS model to the top-down symbolic model that serves as an experimental control, we first examined how varying the symbolic model's decay rate δ_S affected its performance. The results can be seen in Fig. 9. In all three task versions, both the average and the standard deviation of the number of rounds needed to complete the task increased superlinearly with the decay rate. Results from 200 runs of the symbolic model showed the closest fit to human subjects for the $n = 8$, 12, and 16-card conditions when the symbolic decay rate $\delta_S = 0.475, 0.40$, and 0.30, respectively. This is consistent with analogous findings in past computational studies where decreased decay was correlated with increased working memory span (Altmann & Gray, 2002; Reggia et al., 2009; Winder et al., 2009).

Results with GALIS (200 runs for each value of n) are summarized in Fig. 10. These are compared with both the results from human subjects and from three instances of the symbolic model—"perfect play" with no memory decay ($\delta_S = 0$), random play without any memory ($\delta_S = 1$), and intermediate play with $\delta_S = 0.375$, which was the decay value which provided the best fit to the human results across all three task conditions. The mean number of rounds it took GALIS to complete the task increased as n increased, consistent with human performance and with expectations. Using both a Student t-test and a Kolmogorov–Smirnov test, we find no statistically significant difference at the $p = 0.05$ level between the GALIS model's performance and that of human subjects or the best-matching symbolic model on both the 8- and 12-card conditions (of course, a non-statistically significant difference in performance can still be meaningful). The GALIS model performed somewhat worse than humans on the more challenging 16-card version, as did the best-matching symbolic model. In this task condition the neural model slightly out-performed the symbolic model, but not at a significant level.

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

In order to investigate the causes behind the GALIS model's less accurate match to human performance levels under the most challenging task condition where $n = 16$ we constructed histograms of the performance for both humans and the neural model. These, along with a kernel density estimate (KDE) for smoothing, are shown in Fig. 11. As can be seen in the right subplot for $n = 16$, the difference between human and GALIS performance is largely due to a thicker right-hand tail on the distribution of GALIS results. Without these outlying runs (which also occurred with human subjects, but less prominently, and which required over 40 rounds to complete), there was no statistically significant difference from the human results.

Fig. 12 shows a similar plot with the bars omitted for the $n = 16$ variant, with human performance, GALIS results, and results from the symbolic model with two different decay rates. When $\delta_S = 0.35$ there was no significant difference between the performance of the symbolic and GALIS models, and both were worse than the human level. The symbolic model was able to decrease the numbers of rounds needed by lowering its decay rate to 0.3. This slight parameter shift was all that was needed to cause the symbolic model to go from matching the GALIS model to the human performance level for $n = 16$ (but not for other values of n). This indicates a partial cause behind the GALIS model's inability to match human results on this task version: GALIS' decay rate was already set very low, allowing more interference between the increased number of patterns retained early on in its working memory when $n = 16$, and thus impairing its performance.



(a) $n = 12$. (b) $n = 16$.

Fig. 11. Histogram of human (dark gray bars) and GALIS (light gray bars) performance on 12-card and 16-card task versions. Also given is a curve showing a smoothed estimation of each histogram using Gaussian kernel density estimates (KDE). Human results are the solid lines, and GALIS results are the dashed lines.

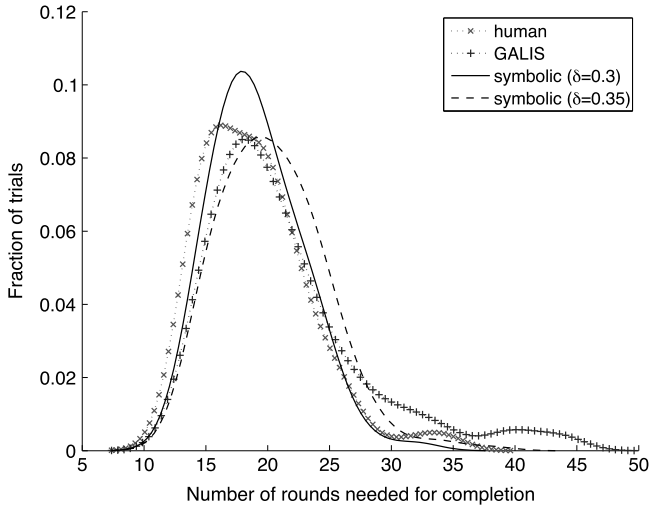


Fig. 12. Kernel density estimates on the 16-card task for human subjects and the GALIS model compared to the symbolic system with two different decay rates. When $\delta_S = 0.3$ the symbolic system and human performance match, and both outperform the GALIS model. Increasing the symbolic system's decay rate to $\delta_S = 0.35$ shifts its performance curve to the right, causing it to be statistically similar to that of the GALIS model. Note also that the performance distribution of the symbolic model displays the same positive skew as do humans and GALIS models, and that the skewness increases with higher decay values.

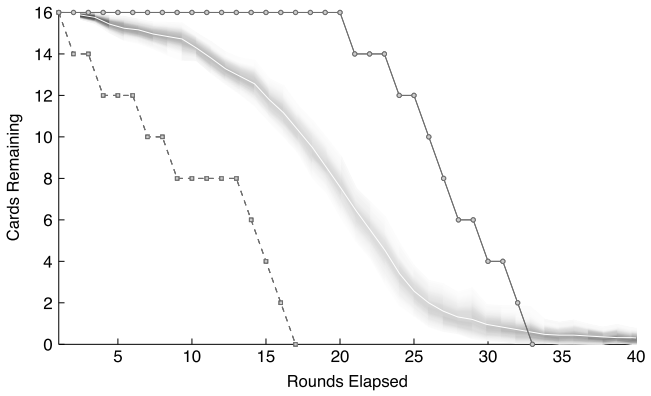


Fig. 13. A visually-weighted regression of the model's average performance over time in 100 runs of the 16-card task variant (gray shading), along with the performance curves of one high-performance run and one low-performance run (lower and upper lines, respectively). The shading represents the width of the confidence interval surrounding the performance, as determined by a nonparametric bootstrap estimate (Hsiang, 2013).

Fig. 13 shows an observation of this pattern. The shaded area in the middle of the plot shows a non-parametric estimate of the average performance of 100 GALIS simulations with $n = 16$ (Hsiang, 2013). Two particular runs of our GALIS model are also shown. The lower line shows one run with a final score of 17, while the simulation represented by the upper line took roughly twice as many turns to finish. The difference is entirely due to the inability of the latter to find the first matching set of cards among the 8 pairs on the board. After this hurdle is cleared the remaining pairs are identified even more rapidly than they are in the high-performing example. This early plateau pattern was characteristic of the few poor-performing simulations that made the GALIS model's results not precisely match those of our human subjects when $n = 16$. Examination of these outlier runs showed that the controller was working precisely as it was trained to do, but that by chance the same location cards were being re-picked frequently early on. The algorithm in Table 1 does not adequately anticipate this possibility, allowing it to occur in a few percent of the simulations and thus biasing the model's performance overall to take a bit longer than humans do in this case.

4. Discussion

The fundamental issue addressed in this work is whether there is an identifiable core set of general-purpose, region-level functions and interactions that can be used to engineer neurocognitive architectures for high-level cognitive tasks. Our hypothesis is that the GALIS framework provides such a set of key functions and interactions: a region-and-pathway architecture inspired by the organization of the human cerebral cortex having neural regions that each serve as attractor networks that are able to learn temporal sequences, and neural regions that not only learn to exchange information but also learn to turn on/off the functions of other regions. The idea of simulated cortical regions that can gate one-another's activations, learning and communications is related to past work that has focused on understanding/modeling cognitive control of working memory via gating operations (Frank et al., 2001; O'Reilly & Frank, 2006; Sherman & Guillery, 2006). Our approach differs from these previous studies in not using biologically-realistic spiking neurons, its focus on generalizing gating to all module functions (activity retention/updating, learning, inter-region activity flow, etc.), and

in allowing cortical regions to directly gate the behavior of other cortical regions without explicitly modeling subcortical brain structures (basal ganglia, thalamic nuclei, etc.).

The concept that one cortical region may directly/indirectly gate the activity of other cortical regions is an especially critical aspect of the work we have described here. While GALIS is agnostic about the potential mechanisms that might implement such gating biologically, this is an issue of much recent and ongoing interest in the cognitive neurosciences. 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). Further, there is substantial evidence that gating occurs in part indirectly 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 oscillatory activity synchronization. Synchronization has been postulated to be an effective way to dynamically gate information flow between cortical areas (Singer, 2011), and this may contribute to top-down attention mechanisms (Womelsdorf & Fries, 2009). A recent review outlines the many ways that oscillatory activity, acting via multiplicative modulation, can direct selective inter-regional communication or "multiplexing" between brain regions (Akam & Kullmann, 2014), something that is consistent with the multiplicative gating implementation used in parts of GALIS systems. 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 the work described in this paper, we have substantially tested our basic hypotheses by applying the GALIS framework to solving challenging card matching problems, a type of problem that is readily addressed by top-down symbolic methods but not by widely used neurocomputational methods. Performing this task requires, among other things, the ability to deal with representing external entities and their locations in space, the ability to support a robust working memory of previously seen cards, the ability to bind together distinct pieces of information about the environment, the ability to interpret and generate sequential events, and the ability to exert top-down attention control and action selection. Such abilities are readily achieved with traditional top-down AI symbolic systems, but have proven to be extremely challenging for neural architectures (Martinet, Sheynikhovich, Benchenane, & Arleo, 2011; Trullier, Wiener, Berthoz, & Meyer, 1997), and go far beyond the modeling of working memory that was attempted with GALIS previously (Sylvester et al., 2013). In ways, our work with the GALIS framework relates to the longstanding idea of "connectionist implementationism", by which we mean the hope that neurocomputational methods will provide a mechanistic account of cognition that goes beyond that provided by more conventional computational approaches (McClelland, Rumelhart, & Hinton, 1986). Our work with GALIS is a step towards showing that neurocomputational systems offer a means for implementing cognitive functions using distributed representations at a subsymbolic level, in particular in capturing the "distinctly sequential character" of cognition that involves state-to-state transitions (McClelland et al., 1986). We believe that our work here contributes something new to this discussion: It demonstrates that the specific postulates of the GALIS framework—that the critical ingredients to implementing cognitive functions are neural modules that learn procedures represented as itinerant attractor sequences, and that these regions should interact not only via the exchange of information but also by gating one another—are viable candidates for aspects of the neurocomputational microstructure underlying the macrostructure of cognition.

The results presented here provide significant support for the GALIS hypothesis that one can engineer high-level problem-solving systems based on regions that are attractor networks and that have the ability to gate the functionality of other regions and pathways. Specifically, we found that such neurocognitive architectures could readily learn to solve card matching problems. Further, the number of steps (card selections) it made during problem solving qualitatively increased with problem difficulty in a fashion similar to that seen when we had a group of human subjects solve this problem. In two of three task conditions our model's performance and the performance we observed with the human subjects we studied matched quantitatively. However, this match deviated in the third condition with the largest number of cards, a significant limitation of our work indicating that our model is not yet fully "human competitive". Why did this happen? We believe that there are multiple potential factors that may come into play as the number of cards increases, as follows.

First, as described in the Results section above, our analysis indicates that one important reason for the deviation was that the author-generated instructions the system was trained to perform were not optimal. The trained system was very accurately executing the "program" it was instructed to perform correctly, but as often occurs with conventional software development, the "code" was not optimal. A second possible factor may simply be the limited size of the working memory and ISM modules in our model when compared to those of the human brain. Size limitations like these can result in increased interference between stored patterns and thus decreased memory capacity. Finally, another intriguing possibility is that some additional unknown human memory mechanisms, not captured in the GALIS framework, could be coming into play as the scale of problems increases. A candidate mechanism might relate to the adaptability of weight decay that influences memory capacity. We examined the influence of working memory decay on our card task system and found that while adjusting the rate was not required in order to qualitatively match the behavior of human subjects, doing so did allow closer fits to the data. This corresponds to past theoretical hypotheses about the role of memory decay on working memory capacity management (Winder et al., 2009). Importantly, from our engineering perspective of wanting to implement high-level problem solving systems with purely neurocomputational mechanisms, none of these limitations are failures of GALIS's underlying principles. Instead they suggest that there are important research issues to be tackled in the future. For example, future research is needed to determine the effects of memory capacity on GALIS systems' performance, to give GALIS systems the ability to modify their originally-learned instruction sequences based on problem solving experiences, and (in cognitive psychology) to gain a better understanding of the relative roles of decay and interference in human memory mechanisms via behavioral experiments.

Our neurocognitive system for card matching is composed entirely of components based on neural network methods that use distributed subsymbolic representations. The finding that this system can perform cognitive problem-solving operations of the sort performed by traditional AI symbolic methods is both encouraging and, we believe, highly significant. Such cognitive control abilities are widely recognized to be challenging for neural computational methods. In a sense, our approach provides a synthesis of continuous neurocomputational representations and symbolic AI representations. Even though our approach uses only neural information processing, the fact that it allows one to "program" a neural

network with a sequence of high-level instructions creates a similarity to the traditional von Neumann architecture computer. Further, even though the attractor networks operate in a high-dimensional, continuous state space, each attractor within that space exists as a discrete entity (Simen & Polk, 2009), and the use of gating allows for hard-cutoff binary distinctions to be made (open vs. closed communication channel, active vs. quiescent region, update working memory vs. maintain its current contents, learn vs. do not learn, etc.). As a result, GALIS networks offer a balance between the continuous nature of neural networks and the discrete nature of symbolic systems. Gating also has the further benefit of providing a way to balance the dual needs of maintaining stability of a network's state and for being able to rapidly switch states (O'Reilly, Noelle, Braver, & Cohen, 2002), a key requirement that has long been recognized as important in biological cognitive control systems (Goldman-Rakic, 1987).

Our work here can be compared to several past related studies of neural systems for working memory and cognitive control. Neural networks have been widely used to model cognitive control (e.g., Botvinick & Plaut, 2006; Kaplan, Sengör, Gürvit, Genç, & Güzeliş, 2006; Pascanu & Jaeger, 2011; Ponzi, 2008; Verduzco-Flores, Ermentrout, & Bodner, 2012). Many of these, such as C-SOB (Lewandowsky & Farrell, 2008), concentrate almost exclusively on the working memory aspect of cognitive control and rely on the modeler to make decisions about when to update weights or how to produce output. Most of the models which incorporate their own cognitive control methods do so by structuring their architecture very specifically to the task at hand. One notable exception is the PBWM architecture (Rougier, Noelle, Braver, Cohen, & O'Reilly, 2005), which is able to iteratively learn rule-like sets of internal representations that guide its pattern classification decisions. Another is the work of Cutsuridis and Taylor (2013), which like GALIS incorporates the use of dual visual pathways for object–location integration. Other very recent work has differed from GALIS in emphasizing the role of reinforcement learning in cognitive control (Zendehrouh, 2015) or in the use of self-organizing maps as a working/episodic memory mechanism (Takac & Knott, 2015).

Our work with GALIS also relates to several past studies done in the cognitive architectures literature. For example, like with symbolic AI, systems such as SOAR (Laird, 2012) and ACT-R (Anderson et al., 2004) have dealt extensively with high-level cognition, working memory, and cognitive control. These issues are primarily and successfully addressed by cognitive architectures via the use of production rules. Our work is complementary to these past studies in that the GALIS framework tries to capture similar functionality using purely neurocomputational mechanisms rather than symbolic production rules. The distinction made between explicit and implicit knowledge in the cognitive architecture CLARION is particularly illuminating in this regard (Sun & Naveh, 2004; Wilson, Sun, & Matthews, 2009). CLARION primarily captures the explicit processing of cognitive control and working memory operations using symbolic rule-based "localist" representations. In contrast, it captures relatively inaccessible implicit knowledge as distributed representations in neural networks. No other system that we know of better illustrates the complementary nature of symbolic and neurocomputational methods that we discuss at the very beginning of this paper. Further, the ideas in GALIS of regions that interact and gate one another are similar in ways to the modules and their interactions in cognitive architectures such as CLARION. From this perspective, our work with the GALIS framework is striving to identify and study the key neurocomputational mechanisms that are needed to capture explicit processing operations using solely neural networks.

While the successful use of the GALIS framework to solve card matching problems is very encouraging, much further work is needed to assess this approach and extend it to even more challenging problems (the current model would even need some extensions to its input and output configurations to be used for the *n*-Back task). It will be important to demonstrate that the methods used here can be adopted to a broader range of problem-solving tasks typical of those used historically in traditional symbolic AI systems. In particular, a key issue for future research is whether the approach adopted in GALIS will scale-up to larger, more complex problem tasks, both in terms of network size and convergence times. Also, since neither sophisticated image processing nor motor control was a focus of this work, expanding such portions of the system to, for example, deal with color and invariance to input transformations, would of course be important future research areas. In addition, "programming" a neural network as we have done here is a fairly new pursuit, and one that would benefit from finding new methods and tools for analyzing the behaviors of large-scale complex network architectures.

Both top-down, symbolic AI and bottom-up, neural systems have achieved impressive results, but each has largely done so in their own separate application domains. A bridge between symbolic and neural approaches would be very advantageous, and the GALIS framework is one way to advance this bridging. To the extent that it and other related research is successful, it may even contribute to a better understanding of the general mind–brain problem (Reggia et al., 2014).

Acknowledgment

This work was supported in part by ONR Grant N000141310597.

References

Abbott, A. (2013). Solving the brain. *Nature*, *499*, 272–274.
Achlioptas, D. (2003). Database-friendly random projections: Johnson-Lindenstrauss with binary coins. *Journal of Computer and System Sciences*, *66*(4), 671–687.
Akam, T., & Kullmann, D. (2014). Oscillatory multiplexing of population codes for selective communication in the mammalian brain. *Nature Reviews Neuroscience*, *15*, 111–122.
Altmann, E. M., & Gray, W. D. (2002). Forgetting to remember: The functional relationship of decay and interference. *Psychological Science*, *13*(1), 27–33.
Anderson, J. R., Bothell, D., Byrne, M. D., Douglass, S., Lebiere, C., & Qin, Y. (2004). An integrated theory of the mind. *Psychological Review*, *111*(4), 1036–1060.
Baizer, J. S., Ungerleider, L. G., & Desimone, R. (1991). Organization of visual inputs to the inferior parietal cortex in macaques. *The Journal of Neuroscience*, *11*(1), 168–190.
Botvinick, M. M., & Plaut, D. C. (2006). Short-term memory for serial order: A recurrent neural network model. *Psychological Review*, *113*(2), 201–233.
Bressler, S., & Menon, V. (2010). Large-scale brain networks in cognition. *Trends in Cognitive Sciences*, *14*, 277–290.
Brown, J., & Braver, T. (2005). Learned predictions of error likelihood in the anterior cingulate cortex. *Science*, *307*, 1118–1121.
Brown, G. D., Preece, T., & Hulme, C. (2000). Oscillator-based memory for serial order. *Psychological Review*, *107*(1), 127–181.
Burgess, P., Dumontheil, I., & Gilbert, S. (2007). The gateway hypothesis of rostral prefrontal cortex (area 10) functions. *Trends in Cognitive Sciences*, *11*(7), 290–298.
Cutsuridis, V., & Taylor, J. G. (2013). A cognitive control architecture for the perception–action cycle in robots and agents. *Cognitive Computation*, *5*(3), 383–395.
de Garis, H., Shuo, C., Goertzel, B., & Ruiting, L. (2010). A world survey of artificial brain projects. *Neurocomputing*, *74*, 3–29.
Eliasmith, C., Stewart, T. C., Choo, X., Bekolay, T., DeWolf, T., Tang, Y., & Rasmussen, D. (2012). A large-scale model of the functioning brain. *Science*, *338*(6111), 1202–1205.
Farrell, S., & Lewandowsky, S. (2002). An endogenous distributed model of ordering in serial recall. *Psychonomic Bulletin & Review*, *9*(1), 59–79.
Feldman, J. (2013). The neural binding problem. *Cognitive Neurodynamics*, *7*(1), 1–11.
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.
Goldman-Rakic, P. S. (1987). Circuitry of primate prefrontal cortex and regulation of behavior by representational memory. In *Handbook of physiology—the nervous system. Vol. 5* (pp. 373–417).
Haykin, S. (2009). *Neural networks*. (pp. 2–6). New York: Prentice Hall.
Heinke, D., & Backhaus, A. (2011). Modelling visual search with the selective attention for identification model. *Cognitive Computation*, *3*, 185–205.

Horn, D., & Opher, I. (1996). Temporal segmentation in a neural dynamic system. *Neural Computation*, *8*(2), 373–389.
Hoshino, O., Usuba, N., Kashimori, Y., & Kambara, T. (1997). Role of itinerancy among attractors as dynamical map in distributed coding scheme. *Neural Networks*, *10*(8), 1375–1390.
Hsiang, S. M. (2013). *Visually-weighted regression. Technical Report SSRN 2265501*.
Ismail, H.O., & Shapiro, S.C. (2000). Two problems with reasoning and acting in time. In *Cohn, A., Giunchiglia, F., and Selman, B., (Eds.), Proc. 7th int'l conf. knowledge representation and reasoning, KR 2000* (pp. 355–365).
Johnson, W. B., & Lindenstrauss, J. (1984). Extensions of Lipschitz mappings into a Hilbert space. *Contemporary Mathematics*, *26*(189–206), 1.
Jones, M., & Polk, T. A. (2002). An attractor network model of serial recall. *Cognitive Systems Research*, *3*(1), 45–55.
Kaplan, G., Sengör, N. S., Gürvit, H., Genç, I., & Güzeliş, C. (2006). A composite neural network model for perseveration and distractibility in the Wisconsin card sorting test. *Neural Networks*, *19*(4), 375–387.
Koechlin, E., & Summerfield, C. (2007). An information theoretical approach to prefrontal executive function. *Trends in Cognitive Sciences*, *11*(6), 229–235.
Laird, J. (2012). *The SOAR cognitive architecture*. MIT Press.
Lewandowsky, S., & Farrell, S. (2008). Short-term memory: New data and a model. *Psychology of Learning and Motivation*, *49*, 1–48.
Lipson, H., & Siegelmann, H. (2000). Clustering irregular shapes using high-order neurons. *Neural Computation*, *12*(10), 2331–2353.
Machens, C. K., Romo, R., & Brody, C. D. (2005). Flexible control of mutual inhibition: A neural model of two-interval discrimination. *Science*, *307*, 1121–1124.
Martinet, L.-E., Sheynikhovich, D., Benchenane, K., & Arleo, A. (2011). Spatial learning and action planning in a prefrontal cortical network model. *PLoS Computational Biology*, *7*(5).
McClelland, J., Rumelhart, D., & Hinton, G. (1986). The appeal of parallel distributed processing. In R. D., & M. J. (Eds.), *Parallel distributed processing* (pp. 3–44). Cambridge MA: MIT Press.
Monner, D., & Reggia, J. (2013). Emergent latent symbol systems in recurrent neural networks. *Connection Science*, *12*, 1932–1943.
O'Reilly, R. C., & Frank, M. J. (2006). Making working memory work: A computational model of learning in the prefrontal cortex and basal ganglia. *Neural Computation*, *18*(2), 283–328.
O'Reilly, R. C., Noelle, D. C., Braver, T. S., & Cohen, J. D. (2002). Prefrontal cortex and dynamic categorization tasks: Representational organization and neuromodulatory control. *Cerebral Cortex*, *12*(3), 246–257.
Owen, A. M., McMillan, K. M., Laird, A. R., & Bullmore, E. (2005). N-back working memory paradigm: A meta-analysis of normative functional neuroimaging studies. *Human Brain Mapping*, *25*(1), 46–59.
Pascanu, R., & Jaeger, H. (2011). A neurodynamical model for working memory. *Neural Networks*, *24*(2), 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*(2–3), 322–330.
Rahimi, A., & Recht, B. (2008). Weighted sums of random kitchen sinks: Replacing minimization with randomization in learning. In *Advances in neural information processing systems* (pp. 1313–1320).
Reggia, J., Monner, D., & Sylvester, J. (2014). The computational explanatory gap. *Journal of Consciousness Studies*, *21*(9), 153–178.
Reggia, J., Sylvester, J., Weems, S., & Bunting, M. (2009). A simple oscillatory short-term memory. In *Biologically inspired cognitive architectures II* (pp. 103–108).
Reynolds, J. H., & Desimone, R. (1999). The role of neural mechanisms of attention in solving the binding problem. *Neuron*, *24*(1), 19–29.
Rougier, N. P., Noelle, D. C., Braver, T. S., Cohen, J. D., & O'Reilly, R. C. (2005). Prefrontal cortex and flexible cognitive control: Rules without symbols. *Proceedings of the National Academy of Sciences of the United States of America*, *102*(20), 7338–7343.
Roy, A. (2008). Connectionism, controllers, and a brain theory. *IEEE Transactions on Systems, Man and Cybernetics, Part A*, *38*, 1434–1441.
Schneider, W., & Chein, J. M. (2003). Controlled & automatic processing: Behavior, theory, and biological mechanisms. *Cognitive Science*, *27*(3), 525–559.
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*(5), 705–761.
Singer, W. (2011). Dynamic formation of functional networks by synchronization. *Neuron*, *69*, 191–193.
Sporns, O. (2011). *Networks of the brain*. MIT Press.
Storkey, A. (1997). Increasing the capacity of a Hopfield network without sacrificing functionality. In *Proc. int'l conf. artificial neural networks, ICANN* (pp. 451–456).
Sun, R., & Naveh, I. (2004). Simulating organizational decision-making using a cognitively realistic agent model. *Journal of Artificial Societies and Social Simulation*, *7*(3).
Sylvester, J., Reggia, J., & Weems, S. (2011). Cognitive control as a gated cortical net. In *Proc. of the 2nd intl'l conf. on biologically-inspired cognitive architectures* (pp. 371–376).
Sylvester, J., Reggia, J., Weems, S., & Bunting, M. (2010a). A temporally asymmetric hebbian network for sequential working memory. In *Salvucci, D. D. and Gunzelmann, G., (Eds.), Proc. of the 10th int'l conf. on cognitive modeling. Philadelphia, PA* (pp. 241–246).
Sylvester, J., Reggia, J., Weems, S., & Bunting, M. (2010b). A temporally asymmetric hebbian network for sequential working memory. In *Salvucci, D. D. and Gunzelmann, G., (Eds.), Proc. 10th int'l conf. cognitive modeling* (pp. 241–246).
Sylvester, J., Reggia, J., Weems, S., & Bunting, M. (2013). Controlling working memory with learned instructions. *Neural Networks*, *41*(0), 23–38.
Takac, M., & Knott, A. (2015). A neural network model of episode representations in working memory. *Cognitive Computation*, *7*, 509–525.
Townsend, J., Keedwell, E., & Galton, A. (2014). Artificial development of biologically plausible neural-symbolic networks. *Cognitive Computation*, *6*(1), 18–34.
Trullier, O., Wiener, S., Berthoz, A., & Meyer, J.-A. (1997). Biologically based artificial navigation systems: Review and prospects. *Progress in Neurobiology*, *51*(5), 483–544.
Ungerleider, L. G., & Haxby, J. V. (1994). 'what' and 'where' in the human brain. *Current Opinion in Neurobiology*, *4*(2), 157–165.
van der Velde, F., & de Kamps, M. (2001). From knowing what to knowing where. *Journal of Cognitive Neuroscience*, *13*(4), 479–491.
van Essen, D. (2005). Corticocortical and thalamocortical information flow in the primate visual system. *Progress in Brain Research*, *149*, 173–185.
van Essen, D., Anderson, C., & Felleman, D. (1992). Information processing in the primate visual systems. *Science*, *255*, 419–423.
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.
Weems, S., & Reggia, J. (2006). Simulating single word processing in the classic aphasia syndromes based on the Wernicke–Lichtheim–Geschwind theory. *Brain and Language*, *98*, 291–309.
Wilson, R., Sun, R., & Matthews, R. (2009). A motivationally-based simulation of performance degradation under pressure. *Neural Networks*, *22*, 502–508.
Winder, R., Cortez, C., Reggia, J., & Tagamets, M. (2007). Functional connectivity in fMRI: A modeling approach for estimation and for relating to local circuits. *NeuroImage*, *34*, 1093–1107.
Winder, R., Reggia, J., Weems, S., & Bunting, M. (2009). An oscillatory Hebbian network model of short-term memory. *Neural Computation*, *21*(3), 741–761.
Womelsdorf, T., & Fries, P. (2009). Selective attention through selective neuronal synchronization. In M. Gazzaniga (Ed.), *The cognitive neurosciences* (pp. 289–302). MIT Press.
Zendehrouh, S. (2015). A new computational account of cognitive control over reinforcement-based decision-making. *Neural Networks*, *71*, 112–123.