Intelligence 42 (2014) 53–82

Contents lists available at ScienceDirect

Intelligence

A spiking neural model applied to the study of human performance and cognitive decline on Raven's Advanced Progressive Matrices

Daniel Rasmussen *, Chris Eliasmith

Centre for Theoretical Neuroscience, University of Waterloo, Waterloo, ON N2J 3G1, Canada

ARTICLE INFO

Article history:
Received 12 April 2013
Received in revised form 19 September 2013
Accepted 21 October 2013
Available online xxxx

Keywords:
Raven's Progressive Matrices
Vector symbolic architectures
Cognitive decline
Aging
Spiking neural model

ABSTRACT

We present a spiking neural model capable of solving a popular test of intelligence, Raven's Advanced Progressive Matrices (RPM). The central features of this model are its ability to dynamically generate the rules needed to solve the RPM and its biologically detailed implementation in spiking neurons. We describe the rule generation processes, and demonstrate the model's ability to use the resulting rules to solve the RPM with similar performance and error patterns to human subjects. Investigating the rules in more detail, we show that they successfully capture abstract patterns in the data, enabling them to generalize to novel matrices. We also show that the same model can be used to solve a separate reasoning task, and demonstrates the expected positive correlation in performance across tasks. Finally, we demonstrate the advantages of the biologically detailed implementation by using the model to connect behavioral and neurophysiological data. Specifically, we investigate two neurophysiological explanations of cognitive decline in aging: neuron loss and representational "dedifferentiation". We show that manipulations to the model that reflect these neurophysiological hypotheses result in performance changes that match observed human behavioral data.

1. Introduction

Cognitive modeling has been making dramatic progress in recent years along two different, but related, dimensions. The first is the modeling of more complex tasks—creating systems that are able to model human performance in an increasingly broad array of cognitively challenging domains (Ashby, Ennis, & Spiering, 2007; Frank & Badre, 2012). The second is the modeling of ever greater biological detail—creating systems that recreate the inner mechanisms of biological brains in increasing fidelity (Bourjaily & Miller, 2011; Gurney, Prescott, & Redgrave, 2001; Hazy, Frank, & O'Reilly, 2007). One of the important challenges now is to connect these dimensions, creating models that are able to perform complex cognitive tasks using biologically detailed mechanisms. These models provide new ways to think about the functional mechanisms giving rise to the complex processes of intelligence, as well as new ways to understand how those mechanisms are connected to the underlying neural processes of the brain.

In this paper we have chosen to explore the Raven's Progressive Matrices intelligence test (RPM; Raven, Raven, & Court, 1991), a reasoning task wherein subjects must induce the rules governing a set of geometric figures in order to complete an often complicated pattern. The primary reason for choosing this task is its widespread usage in both clinical and research settings. This ensures that there is an abundance of neurophysiological and behavioral data, which we can use both to constrain the model and to evaluate its performance. This will enable us to examine whether or not the model succeeds in the dual goals of accounting for complex cognitive performance with biologically detailed

* Corresponding author.
E-mail addresses: drasmuss@uwaterloo.ca (D. Rasmussen), celiasmith@uwaterloo.ca (C. Eliasmith).

0160-2896/$ – see front matter © 2013 Elsevier Inc. All rights reserved.
http://dx.doi.org/10.1016/j.intell.2013.10.003

mechanisms. Previous work has described a large-scale model that included a simplified version of the system presented here (Eliasmith et al., 2012). In that work the focus was on integrating sensory input, reasoning, and motor output; here we focus on the reasoning system in detail, both greatly expanding its function and providing a detailed analysis of its performance.

Along the cognitive dimension, the model we present here is unique in its ability to generate the rules describing a Raven's matrix in a dynamic, flexible, general manner. Previous models have relied on rules built in by the modelers, or rule systems tailored to the Raven's test. While these models provide interesting insight into various aspects of Raven's performance, it seems likely that human subjects employ more general rule generation abilities. For example, analysis of the RPM itself (Marshalek, Lohman, & Snow, 1983), as well as psychometric and neuroimaging practice (Gray, Chabris, & Braver, 2003; Perfetti et al., 2009; Prabhakaran, Smith, Desmond, Glover, & Gabrieli, 1997), show the RPM to be a fluid task with a minimal dependence on previous experience. With the proposed model we provide an explanation for how rules could be generated in the dynamic, general manner exhibited by human subjects.

A second novel aspect of this model is its biologically detailed implementation. The presented model consists of networks of spiking neurons modeled on the physiology, connectivity, and dynamics of biological neurons. This connection to biology provides several advantages. One is that it allows us to use neurophysiological evidence to provide constraints on the model, ensuring that its mechanisms operate successfully within the limitations and abilities of the human brain. Another is that the neural implementation allows us to map the results of the model onto a wider variety of empirical data. For example, we can compare neurophysiological changes observed in humans to variations in the neural mechanisms of the model. More symbolic models require an additional layer of abstraction—reinterpreting neurophysiological changes in terms of non-neural variables such as signal noise or response time.

In order to demonstrate this feature of the model, we apply it to the problem of understanding the causes of cognitive decline. This is a clear example of the type of problem that requires the melding of complex cognitive performance and biological detail. As we get older, we decline in a broad range of behavioral measures, from simple processing speed (Salthouse, 1996) to overall performance on batteries of intelligence tests (Kaufman, Reynolds, & McLean, 1989). More recently there has arisen a growing body of evidence describing changes in neurophysiological variables that might account for these declines, such as gray matter reduction (Fjell et al., 2006)—which could be associated with neuron death, neuron atrophy, or dendritic shrinkage—myelin damage (Peters & Sethares, 2002; Sullivan, Adalsteinsson, & Pfefferbaum, 2006), loss of connectivity (Goh, 2011; Madden, Bennett, & Song, 2009), dedifferentiation of neural representations (Park, Carp, Hebrank, Park, & Polk, 2010; Park et al., 2004), and reduced neurotransmitter efficiency (Kaasinen et al., 2000). However, the challenge in testing such hypotheses is that when investigating these alternatives in elderly subjects, most if not all of these factors are present to some degree. This makes it difficult to isolate the effects of any one factor, or to investigate systematically how they interact. In addition, researchers are restricted to observing the levels of these factors available in their subjects; in general, they cannot directly manipulate the variables in which they are interested. Our results help to demonstrate how biologically detailed cognitive neural modeling can begin to address these challenges.

We begin by describing the model and its methods. We then explore the first dimension, complex cognitive performance, providing several demonstrations of the model's ability to generate appropriate rules that enable it to solve Raven's matrices. Next, we explore the second dimension by using the biologically detailed implementation to examine two hypothesized neurophysiological aging factors: neuron loss and representational "dedifferentiation". We conclude with a discussion of the advantages and disadvantages of the model, and of the general principle of creating models that combine complex cognitive performance with a biologically detailed implementation.

2. Background

2.1. Raven's Progressive Matrices

The Raven's Progressive Matrices test (RPM) was originally developed in 1936. Since then it has undergone several revisions, new versions have been added, and it has become one of the most widely used tests of general intelligence (Van De Vijver, 1997). The version of the test used in our research is the Advanced Progressive Matrices (unless otherwise mentioned, when we refer to Raven's matrices or the RPM we are referring to the Advanced version). The two other versions, Colored and Standard, are generally used to test children or lower performing adults (such as those with cognitive deficits), while the Advanced test is used to differentiate average/above-average adult subjects. Since we are interested in understanding normal, unimpaired adult reasoning, the Advanced test is most appropriate for studying those subjects. It also represents the most difficult version of the test, presenting the greatest challenge for the goal of combining complex cognition with biological detail.

Each question in the RPM consists of a single matrix. Each matrix consists of a 3×3 arrangement of cells, and each cell contains an assortment of geometric features, with the exception of a blank cell in the bottom right (Fig. 1).[1] There are eight candidate answers for the blank cell given underneath the matrix. A matrix is governed by a set of rules that describe patterns along the rows or columns of the matrix. The subject's task is to induce those rules based on the information presented, and then determine which of the eight possible candidates would best complete the pattern for the third row/column if placed in the blank cell.

2.1.1. Previous models

The most influential model of the RPM is that of Carpenter, Just, and Shell (1990). Theirs is a production system (Concurrent, Activation-based Production System; Thibadeau, Just, & Carpenter, 1982) based model; it operates by moving information in and out of a shared memory area, and the presence of certain patterns of information in that memory can trigger the execution of different "productions",

[1] To preserve the security of the test we present modified matrices in the figures of this paper; the model works with the true Raven's matrices.

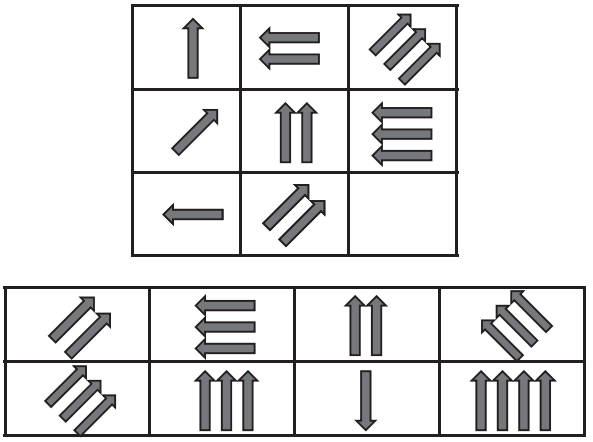

Fig. 1. A simple Raven's-style matrix. The number of arrows is increasing by one along the row, and in each row the arrows are pointed upwards, to the left, and to the right (with the order shuffled). The correct answer is three arrows pointing upwards.

which further modify the contents of the memory. Its solution process is based on detecting smaller patterns in the matrix, building those patterns up into complete rules, and then applying those rules to select an answer. This was the first model able to solve Raven's matrices, and it revealed the impressive ability of modeling to provide new insight into what was already a well-studied task at that time. Carpenter et al. used their model to demonstrate the importance of working memory in RPM performance, as well as provide support for qualitative differences in RPM rule types. There are two main weaknesses to this model, however. First, its rule system is closely tied to the structure and content of the RPM. It takes a "recognition" based approach, detecting externally defined patterns that characterize common Raven's rules (such as an increase in the number of objects with the same shape), rather than building generic rules in a bottom-up fashion from the matrix data. Second, the model is characterized only at a high level of abstraction. It operates in a purely symbolic fashion, and it is unclear how the rules and structures of the model map onto lower level (such as neuroanatomical or neurophysiological) processes. Although this still allows many interesting comparisons to human performance, there are some questions that cannot be answered with such a model. For example, their model simulates changes in working memory ability by placing fixed limits on the number of rules that can be stored in memory (3, 4, 5, or unlimited). By providing a biologically detailed explanation, our model is able to demonstrate a more natural gradient of performance, as well as enable analysis as to the neurophysiological changes that might drive those changes in storage ability (see Appendix C). However, this should not be taken to understate the importance of the Carpenter et al. (1990) model; it laid the groundwork for all future modeling on this task, and still shapes much of our understanding of RPM performance.

The model of Lovett, Forbus, and Usher (2010) was developed for the Standard Progressive Matrices as opposed to the Advanced version used in the Carpenter et al. (1990) model and our own; however, there are many similarities between the two test versions, and so we include a discussion of their model here. The Lovett et al. model makes interesting steps in automating the visual processing of the matrices. The matrix must be manually segmented into its component shapes, but the model then uses an automated sketch recognition system to extract relationship information among those components (e.g., "inside of", "next to"). Next, the model applies Gentner's Structure Mapping Engine (1983) to those representations in order to identify corresponding elements across the rows. The model then examines the corresponding elements to identify which of several predefined geometric transformation rules describe how those objects change across the row (Identity, Transformation, Deformation, Shape Change, or Addition/Removal). The main difference between the Lovett et al. model and the one presented here is that they have focused their efforts on visual processing—extracting spatial relationships and identifying corresponding elements—but still rely on rules defined by the modelers. Our model has the opposite focus; we do not include visual processing, but account for the dynamic generation of rules. In addition, the Lovett et al. model is more computational than biological; there is little indication how it might be implemented in the neural hardware of the brain. This limits the insight the model can provide into human problem solving processes. As an example, because of its primarily computational approach the model is completely deterministic, producing the same answers, the same rules, and the same overall score each time it is run. This makes it difficult to explore individual differences or other factors that may increase or decrease performance. However, the effort to automate visual processing is an important one, and the Lovett et al. model makes interesting progress in that direction.

The work of Kunda, McGreggor, and Goel (2012) takes a novel approach, doing away with propositional representations entirely—they work purely with the visual information in the matrix image. They present two related models, referred to as the fractal and affine models. The models look at either whole cells (affine) or small segments (fractal), such as a square of pixels, and try to determine if a known transformation can be used to map that image data onto another cell. The set of all the transformations for the data in a cell represents the "rule" for that cell (i.e., it describes how one cell can be transformed to generate another). Even more-so than the Lovett et al. model, this is a computationally rather than biologically inspired solution to the Raven's matrices problem. This is not to say that there are not interesting insights to be made from the model's general performance (for example, the authors have used their model to examine the error patterns of autistic subjects), as well as the ability to solve the Raven's problems using only pixel-based representations. But if the goal is to understand the underlying mechanisms the human brain might use to solve these tasks—and the intricacies of human performance that arise from those mechanisms—then a new approach is required.

2.2. Modeling methods

There are two theoretical frameworks that form the basis of our model, each corresponding to a different level of representation. The first, more abstract, level manipulates the information contained in a Raven's matrix using psychologically relevant mathematical operations. This is characterized using a Vector

Symbolic Architecture (VSA). The second, neurally grounded, level is concerned with implementing those mathematical representations and operations in large populations and networks of realistic simulated neurons. For this we use the Neural Engineering Framework (NEF). We give a brief overview of the theory behind VSAs here; for a similar discussion of the NEF, see Appendix A.

2.2.1. Vector symbolic architectures

VSAs are a family of techniques that use high-dimensional vectors to represent structured, conceptual information (Gayler, 2003). For example, in the sentence "the dog chases the ball" there are three basic semantic elements—"dog", "ball", and "chase"—that sit in a particular syntactic relation. When translating such a sentence into a mathematical representation, it is necessary to represent both the items and those relations—"the dog chases the ball" is very different from "the ball chases the dog". VSAs allow us to achieve the goal of representing the individual elements as well as their role in the structure.

First we construct a vocabulary of semantic elements, such as "dog", "ball", "chase", and so on. This is done by associating a high-dimensional vector with each word. For example, "dog" could be [0.3 0.4 0.1 …], "cat" could be [0.2 0.6 0.4 …] and "chase" could be [0.7 0.2 0.1 …]. For present purposes, the specific values of these vectors are not important, all that matters is that each vector corresponds to a concept, and that it is possible to identify the concept being represented by examining the vector.[2]

An important question when considering a vocabulary is how many words can be distinguished. In VSAs, this amounts to asking how many unique points can be represented by the vectors. Theoretically, given perfect precision, the answer is that even a one-dimensional vector can represent an infinite number of concepts, because we can simply keep picking a different value for that vector and associating it with a new concept. However, given the approximation present in the VSA operations (as we will see), as well as the inherent noise and imprecision of biologically plausible neurons, the practical number of concepts that can be represented has limits. For example, if we return to a one-dimensional vector with values ranging between 0 and 1, and assume that vectors must be at least 0.1 apart in value in order to be recognizable, then the vocabulary can represent at most 11 distinct "words" ([0.0], [0.1], …, [1.0]).

Despite these limitations, VSAs do scale up well to large vocabularies. Fig. 2 shows an example of this, demonstrating how the number of dimensions required increases with the number of words we want to represent.[3] The particular relationship between vector dimension and vocabulary size depends on the similarity we allow between the vectors. If we allow vectors to be at most 50% similar to each other then we can fit more vectors into the vocabulary than if we only allow 30% similarity. There is no fixed value for that percentage; depending on the noise and precision required in a particular task, it might be possible to distinguish vectors with 10% accuracy or 90%. The important point is that as the number of words we want to represent increases, the required dimension increases in a sublinear fashion. In other words, the size of the vectors does not explode for large vocabularies. For a more in-depth analysis of the biological plausibility of implementing adult-sized vocabularies in VSAs, see Stewart, Tang, and Eliasmith (2011) and Eliasmith (2013).

Having characterized a vocabulary representation, we must also combine the elementary vectors (words) together to represent structure. For this, VSAs require two operations: a superposition operation that combines vectors into a set, and a binding operation that "ties" two vectors together. The important aspect of the first operation is that it creates a new vector that is similar to each of its inputs (where similarity is defined as the inner product of the two vectors). The important aspect of the second operation is that it creates a new vector that is different from each of its inputs, but from which those original inputs can still be recovered. Different VSA implementations are defined by their choice of these operators.

We follow the Holographic Reduced Representation implementation (Plate, 2003) in using vector addition as our superposition operation and circular convolution as our binding operation.[4] Vector addition is the element-wise addition of the two vectors, which can be thought of as producing an average of the two. Circular convolution is more complicated, defined as

$$C = A \circledast B$$

where

$$c_j = \sum_{k=0}^{n-1} a_k b_{j-k \bmod n}. \tag{1}$$

Circular convolution can be thought of as the multiplication to superposition's addition (e.g., it is commutative, associative, and distributive). Circular convolution meets the two criteria of a binding operation: it produces a vector which is not similar to A or B, yet it is still possible to recover A or B from C. Recovery can be accomplished using a pseudoinverse. The pseudoinverse of A, A' is defined as

$$a_i' = a_{-i \bmod n}. \tag{2}$$

This results in the flipping of all elements except the first (e.g. [1 2 3 … 10] becomes [1 10 … 3 2]). The pseudoinverse

[2] Note that just because it is not necessary for the vectors to have anything other than a random relationship to the concepts they represent does not mean that we could not create a non-random relationship if we chose. This idea is explored in detail in Eliasmith (2013).

[3] These values were calculated by randomly generating sets of vectors of the given vocabulary size and an initially low dimension. 100 attempts were simulated to generate a vocabulary where no two vectors in the vocabulary exceeded the given threshold (similarity is calculated as the inner product of the two vectors). If no successfully distinct vocabularies were found, the dimension was increased and the process was repeated. Thus these values represent a rough approximation only, and overestimate the true minimum dimension required; for a more thorough analysis of vocabulary accuracy see Plate (2003).

[4] In true HRRs all vectors and operations on those vectors are normalized. However, sometimes in our model we relax this constraint in order to simplify the computations, settling for approximate normalization. Thus this is not a true HRR implementation, but the approximation does not significantly alter the representation, given the noise already inherent in neural representations.

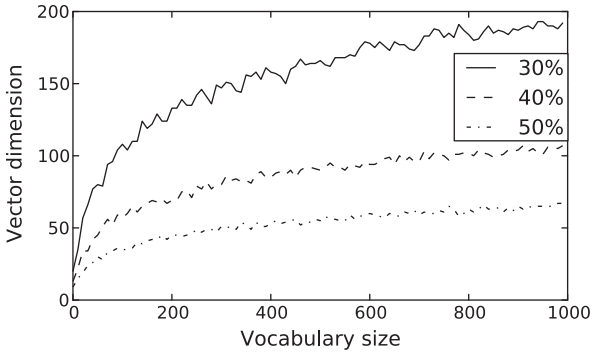

Fig. 2. Minimum vector dimension required to ensure that no two vectors in the vocabulary exceed 30%, 40%, or 50% similarity.

has the useful property that $A \circledast A' \approx I$, where I is the identity vector. This can be used as follows:

$$\begin{aligned} C &= A \circledast B \\ C \circledast B' &= A \circledast B \circledast B' \\ &\approx A \circledast I \\ &\approx A. \end{aligned} \tag{3}$$

In other words, given the vector C containing A and B, it is possible to recover A by convolving C with the inverse of B (and vice versa to recover B).

With these tools it is possible to encode structured information. For example, to encode "the dog chases the ball" we could create a vector $A = \textit{subject} \circledast \textit{dog} + \textit{object} \circledast \textit{ball} + \textit{verb} \circledast \textit{chase}$ (*subject*, *object*, and *verb* are just more vocabulary vectors). This is different from "the ball chases the dog" because $\textit{subject} \circledast \textit{dog}$ is different from $\textit{subject} \circledast \textit{ball}$; we are not only encoding what elements are present, but the structure of those elements—how they are related to one another. We can also encode hierarchical structure, such as combining an independent clause (*ic*) and a dependent clause (*dc*) in the phrase "the dog chases the ball while at the park". We might encode this as $C = ic \circledast A + dc \circledast B$ (where B represents "while at the park").

The information can also be extracted back from the combined vector. For example, if we had C and wanted to know the independent clause, we would calculate

$$\begin{aligned} C &= ic \circledast A + dc \circledast B \\ C \circledast ic' &= (ic \circledast A + dc \circledast B) \circledast ic' \\ &= ic \circledast A \circledast ic' + dc \circledast B \circledast ic' \\ &\approx A \circledast I + noise \\ &\approx A. \end{aligned}$$

(The noise term results from the fact that convolving arbitrary vectors results in new vectors dissimilar to anything in the vocabulary, meaning they can be treated as noise.) We could then use A in further computations. For example, we could determine the subject by calculating $A \circledast \textit{subject}' \approx \textit{dog}$, or we could determine what role "dog" held in the sentence by calculating $A \circledast \textit{dog}' \approx \textit{subject}$.

There are many other encoding schemes that could be used to describe the structure of a sentence, and VSAs can be used to represent any type of structured information, not just sentences. VSAs are simply a general tool for representing and manipulating information—the particular implementation of those representations can be suited to the task at hand. In our model we use an *attribute* ⊛ *value* encoding scheme (with some variation due to the structure of the problems). For example, we could encode $cell_{1,1}$ of Fig. 1 as $\textit{shape} \circledast \textit{arrow} + \textit{number} \circledast \textit{one} + \textit{angle} \circledast 90°$. Carpenter et al. (1990) use a similar attribute–value scheme to encode the matrices, although in their case the attributes and values are symbols rather than vectors. We discuss the details of how we use VSAs to represent Raven's matrices in Appendix B.

3. Model description

There are two primary contributions of this model. First, it provides a theoretical account of rule generation that works in a flexible, general fashion. Second, it demonstrates that this theoretical account can be implemented in a biologically detailed neural simulation. In this section we describe how that dynamic rule generation is accomplished. For a more detailed discussion of the neural implementation, see Appendix A.

3.1. Scope

With these goals in mind, it is also helpful to be explicit about which problems this model is not attempting to solve. To aid in this, we can roughly break the problem of solving a Raven's matrix down into several subtasks. When solving a matrix, such as the one shown in Fig. 3, the subject needs to:

1. parse the image into its component parts; for example, they need to recognize that there is a big empty arrow to the right of a smaller shaded circle in the top left cell.
2. guess which objects correspond to each other (i.e., which objects are they going to try to find a rule for). For example, in this matrix it is natural to think that the big, empty arrows correspond to each other, and the small, shaded circles correspond to each other.
3. find a rule for the corresponding items; if they were looking at the shaded circles, they might notice that the number of circles is increasing by one across each row.
4. use the rules to select an answer—for example, they might generate a hypothesis as to what the blank cell should look like based on their rules, and then choose the answer that is closest to their hypothesis.

Our model is centered on the third component, rule generation. Answer selection is also included so that we can evaluate whether or not the model has discovered correct rules, but is not the focus of the model. The visual parsing and correspondence finding are not part of our model; those steps are accomplished when the matrices are translated from visual into VSA format, which is done systematically but not by the model itself (see Appendix B). In previous work we have investigated the question of integrated performance, showing how a simplified version of this model can be embedded within a complete system combining visual input, reasoning, and motor output (Eliasmith et al., 2012). However, current models of visual processing are fairly limited, which prevents them from being applied to complex visual problems such as the RPM; for example, one of the main limitations of the above model is that the vision system

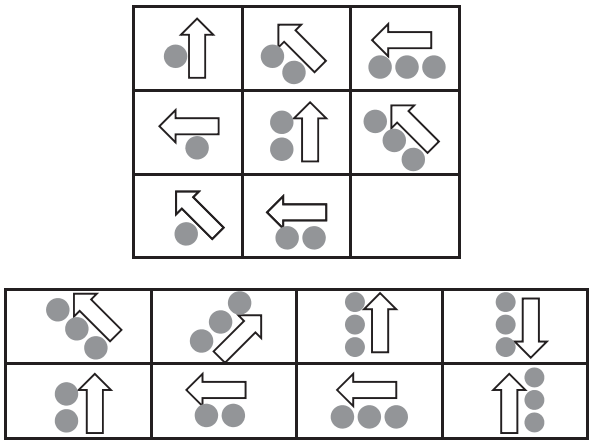

Fig. 3. A simple Raven's-style matrix, illustrating the different aspects of matrix problem solving.

is only able to recognize the digits 1–9 and a few letters. Here we instead focus on the reasoning (rule generation) system in detail, which enables the demonstration and analysis of much more complex cognition.

This raises the question of whether it is possible to study these systems in isolation, or whether any model without a complete explanation from sensory input through to motor output is incomplete. This is certainly an important issue; the rule generation is impacted by these other systems, so our model of rule generation cannot be completely accurate without those components. But conversely, rule generation is an integral part of solving Raven's matrices, and so a greater understanding of that component enhances our understanding of the whole. We will demonstrate in Section 4 that this model is able to account for many aspects of human performance, and so believe that it still provides useful insight into human cognition in this task. One advantage of our modeling approach is that everything is built on the common language of a spiking neural architecture (in our case, the NEF). The NEF is not specific to any area of the brain or aspect of cognition, it is a general tool for representing and manipulating information using neurons. This means that as different systems are developed in the future using spiking neural architectures (e.g., a vision system capable of recognizing geometric shapes), they can be integrated with one another into more comprehensive models.

3.2. Model architecture

A general picture of the model's structure is shown in Fig. 4. The rule generation systems are the three components shown along the bottom. Each one is implemented in neurons (around 20,000 in total across the components), and takes the matrix in VSA form as input and returns the rules it has generated as well as its hypothesis as to the contents of the blank cell. The control structure is responsible for coordinating these systems, inputting the matrix data, and outputting the model's final response.

The three rule generation components attempt to generate different types of rules, where the types are distinguished by the information that constitutes a rule. In this work we describe rule generation in terms of Raven's matrices, but these rule types represent general processes that could be applied to many different types of problems. Matrices containing the three types of rules are shown in Fig. 5. "Sequence" rules involve an orderly progression (e.g., increasing in number, rotation, scaling). These sequences are defined by an iterative transformation applied to the previous item in the sequence; for example, the sequence 4, 5, 6 is defined by the operation $+1$ ($4+1=5, 5+1=6$). The rule that the sequence component generates is that iterative transformation (i.e., given the sequence 4, 5, 6, it will return a rule equivalent to $+1$). "Set" rules involve a set of features that are repeatedly presented, usually with their order being shuffled. In this case the rule being generated is the items that make up the set. "Operation" rules involve two items being combined in some way to form a third; what needs to be learned is the operation being used to combine the two items. For example, given three cells A, B, and C, the operation component will try to generate an operation ($\diamond$) that satisfies the problem $A \diamond B = C$. Note that the rule types are not mutually exclusive—some matrices may be solved correctly by multiple methods. However, all three are required to completely describe the types of patterns found in the RPM.

From a modeling perspective, we employ three different components to find these rules because they represent three fundamentally different types of computations. However, these distinctions are also supported by empirical work on the RPM. Several studies have found qualitatively different problem types and associated problem solving strategies. The most common way that these are described is visuospatial problems (which would encompass our operation and sequence rules) and analytic problems (our set rules). Kirby (1983) found empirical evidence for this distinction in human subjects by creating matrices with two "correct" answers—one if the subject employs a visuospatial rule, and another if the subject is using an analytic rule. He showed that he could cause subjects to use one strategy or the other depending on the verbal instructions used while administering the test and the order in which the items were presented. This demonstrates that humans can use at least two different types of reasoning to solve matrix problems, and switch between them. Carpenter et al. (1990) created a similar rule taxonomy based on verbalization and eye tracking studies. Their system had five rule types, which fall within our general components: figure addition or subtraction (captured by our operation component), distribution of two values or distribution of three values (captured by the set component), and quantitative pairwise progression or constant in a row (captured by our sequence component).

There is also neuroimaging evidence that different problem types lead to relatively enhanced activity in different brain regions. Prabhakaran et al. (1997) found that, broadly speaking, visuospatial RPM problems show predominantly right hemispheric activations (in particular right middle frontal gyrus), while analytic problems showed a bilateral or left-dominant activation. More recently, Golde, von Cramon, and Schubotz (2010) investigated sequence versus set type problems (they did not include any operation problems), and found that again there were consistent differences in the patterns of activation based on the type of problem being solved. In general, these

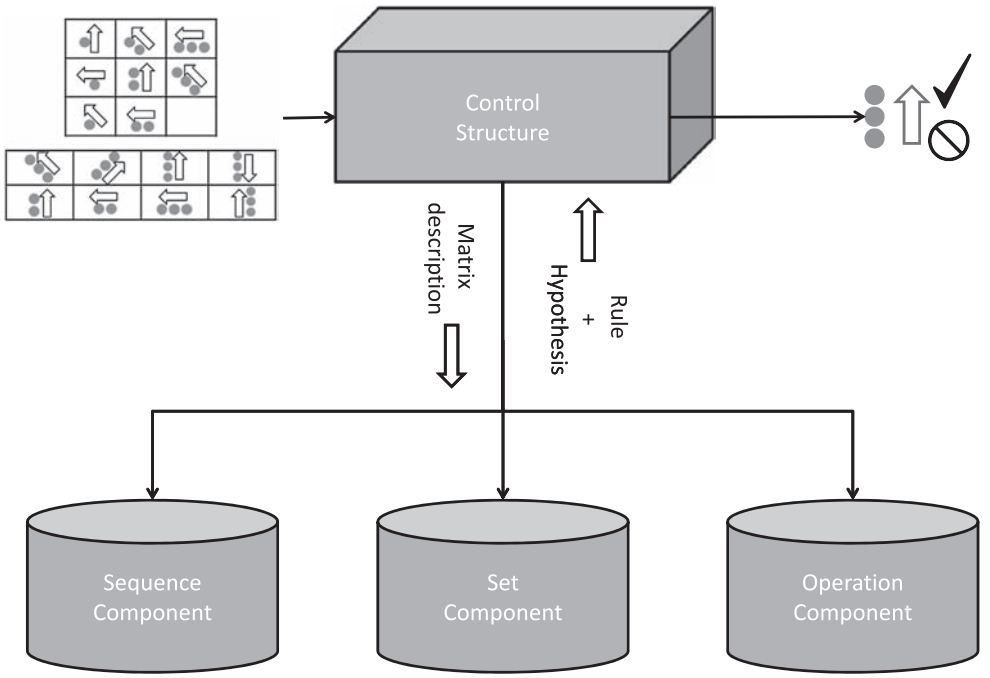

Fig. 4. Overall architecture of the model. The three neural components are displayed along the bottom. Each component takes a description of the Raven's matrix in VSA form as input, and returns the rules that describe that matrix and a hypothesis for the contents of the blank cell. The control structure coordinates the neural components, providing input and collecting output in order to generate a final answer, which it checks for correctness to gather statistics on the model's performance.

studies show that the degree and type of distinction we have made is consistent with the behavioral and neurophysiological evidence.

Given the neural basis of this model, it is desirable to map the components of the model onto specific neuroanatomical areas. However, given current neuroimaging evidence we can do this only in a very general sense. The problem lies in the fact that complex tasks such as the RPM elicit widespread activation throughout the brain, making it difficult to separate out different functional components, such as rule generation.

In general, RPM activations fall into a broad frontoparietal network common in complex reasoning tasks (for a review, see Jung & Haier, 2007). Prefrontal regions have long been associated with high fluid intelligence (Duncan, Burgess, & Emslie, 1995; Duncan & Owen, 2000), the construct most often associated with the RPM. Rule generation in particular seems to be dominated by activation in the prefrontal cortex, especially dorsolateral regions (DLPFC); Seger and Cincotta (2006) found activation associated with the learning of new rules to be primarily found in prefrontal areas, and Golde et al. (2010), Kroger et al. (2002), and Christoff et al. (2001) all found that activation in DLPFC was correlated with the number of rules required to solve the matrix. Thus it is in this region that we believe the model's rule generation components lie.

We posit the control structure to be centered around the basal ganglia. This has been shown to be involved in action selection by Redgrave, Prescott, and Gurney (1999), and to be active during matrix reasoning (Christoff et al., 2001; Melrose, Poulin, & Stern, 2007; Seger & Cincotta, 2006). In addition, Stewart, Choo, and Eliasmith (2010) have shown that a realistic neural model of the basal ganglia is able to carry out the types of control tasks employed in our model.

Parietal brain areas also feature prominently in neuroimaging studies of RPM performance (e.g., Lee et al., 2006), but we believe the role they play lies largely outside this model. Bor, Duncan, Wiseman, and Owen (2003) suggest that while the primary processing occurs in DLPFC, parietal areas are more involved in storage; this is important for RPM performance, both in storing the input information and the results before response selection, but those functions are performed by the controller in our system and therefore not modeled at the neural level. Another role commonly attributed to parietal areas is encoding visual input into a symbolic form (Jung & Haier, 2007), another task that is crucial to RPM performance but outside the current model.

Overall, this localization is very coarse, but as more accurate and fine-grained neuroimaging data becomes available for these complex tasks, we will be able to develop increasingly detailed mappings between neural models and the underlying neuroanatomy.

We now discuss each of the components in turn, examining how the rules are induced. For a discussion of how the computations are implemented at the neural level, see Appendix A.

3.3. Sequence component

The sequence component is designed to find the iterative transformations that define a sequence of items. In the context of Raven's matrices, these transformations could be incremental

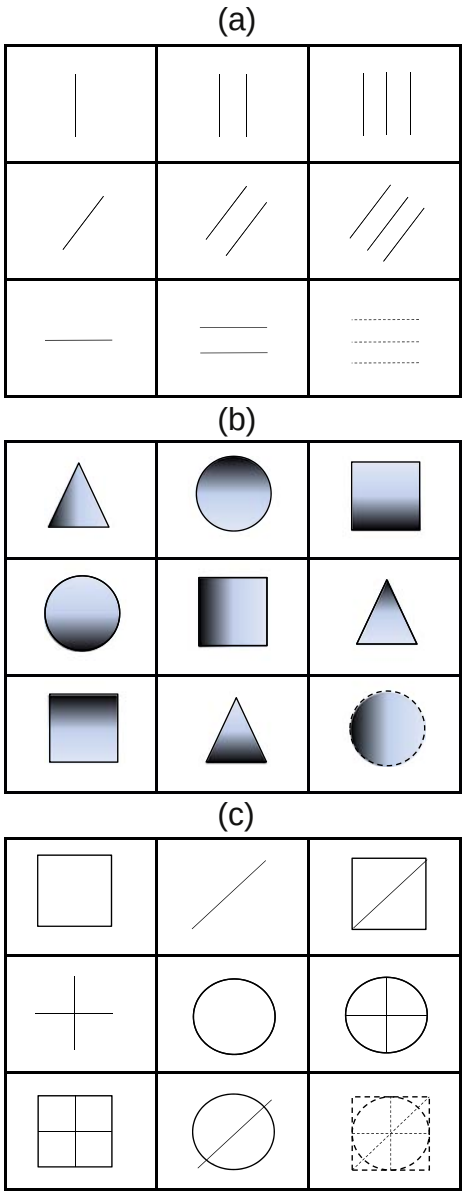

Fig. 5. Sample matrices containing examples of the three different types of rules distinguished by the model. See text for details.

rotations, scalings, translations, or increases/decreases in number; the key aspect is that there is a single manipulation that is being repeatedly applied to generate the next item in the sequence. In Fig. 5(a) there are two sequence rules in play: the number of lines is increasing by one, and the angle of the lines is constant (constant can be thought of as a sequence defined by the identity transformation).[5]

[5] Alternatively, we could look at the columns of the matrix as following a set rule, with the orientation of the lines switching between three different possibilities. However, Carpenter et al. (1990) found in eye-tracking and verbalization studies that subjects solving the RPM predominantly use horizontal, row-based rules to describe the matrices, and so we similarly restrict the model to row-wise descriptions.

Because we are working with a matrix represented in VSA form, this is equivalent to saying that there is a single transformation being applied to one vector (e.g., the top left cell, $cell_{1,1}$, of the Raven's matrix) to generate the next vector in the sequence ($cell_{1,2}$). We know that the operation is circular convolution, so the question is what vector is being convolved with $cell_{1,1}$ to give $cell_{1,2}$. If we call that vector T, then we want to solve for T where $cell_{1,1} \circledast T = cell_{1,2}$, or equivalently $T = cell_{1,1}' \circledast cell_{1,2}$. For example, if $cell_{1,1}$ was one triangle and $cell_{1,2}$ was two triangles, when we calculate T the resulting vector will be one which when convolved with the vector for one triangle gives (approximately) the vector for two triangles. Thus the T vector is a sequence rule—it says how to generate the second item given the first.

However, we do not just want to calculate T for a single pair of cells, we want to calculate a general rule for the whole matrix. To accomplish this we calculate an individual transformation for each adjacent pair of cells in the matrix, and then average those transformations to extract the general rule. For example, if we look at all the individual transformations for a matrix, there may be one for moving from “one triangle” to “two triangles”, one for moving from “two triangles” to “three triangles”, one for moving from “one square” to “two squares”, and so on. But when we average across those transformations, what will be left is what all of those transformation vectors had in common: the general transformation of increasing the number of shapes by one. This is the sequence rule for the matrix, the iterative operation that can be repeatedly applied to any item to generate the next item in the sequence. We can use this rule to solve the matrix by applying it to the second last cell of the matrix ($cell_{3,2}$), because the next item in that sequence should be whatever belongs in the blank cell.

The structure of this component is shown in Fig. 6. The inputs, A_i and B_i, are two adjacent cells of the matrix (the inputs will cycle across the different pairs of cells). The first population calculates the transformation vector between those two cells (the circular convolution of the pseudoinverse of A_i, A_i', and B_i). The second population calculates the running average of the individual transformations (see Fig. A2). The final population generates a prediction for the blank cell ($cell_{3,3}$) by convolving the second-last cell with the average transformation. The neural implementation of this component is described in Appendix A.

3.4. Set component

The set component looks for rules involving repeated sets of items. In the RPM, these could be sets of shapes (e.g., one square, one circle, and one triangle in each row) or more subtle sets of attributes (e.g., one shape pointing to the left, one shape pointing to the right, and one pointing straight up). The order of the items tends to be shuffled in each row, but that is not a necessary part of these rules. In Fig. 5(b) there are two sets in play: one for the shape (triangle, circle, square) and one for the shading (left, top, bottom).

VSAs provide a natural method to represent sets, through the use of the superposition operation. Recall that the superposition of two vectors creates a new vector that is somewhat similar to each of the inputs. Thus if we superimpose multiple vectors we will have a vector that is somewhat similar to all the vectors contained within it, which can be thought of

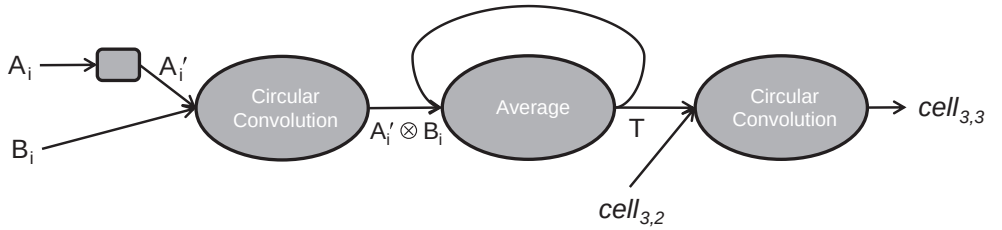

Fig. 6. Schematic diagram of the sequence component. The first population calculates the circular convolution of the two inputs, representing the transformation between those two cells. The second population calculates the average transformation as the inputs cycle through the different pairs of cells in the matrix. The third population calculates the circular convolution of the transformation and the second last cell, representing the prediction for the blank cell.

as representing the set of those items. That is how the set component calculates its rules. It superimposes the items in each set (in the case of the RPM, the cells in one row), and then averages the different set vectors (corresponding to different rows) to generate a general set representation for the matrix. To generate an answer, it uses a process of elimination: it subtracts the known information in the incomplete set (the third row) from the general set, and the remaining vector is the component's hypothesis for the blank cell.

The structure of this component is shown in Fig. 7. The three inputs, A_i, B_i, and C_i, are the three cells in a row. These inputs are summed to generate a vector representing the set of items in the row. The "Average" population calculates the average of the set vectors for each row (see Fig. A2). The final population sums the average set vector with the negative representation of the first two cells in the third row, giving a prediction for the blank cell. The neural implementation of this component is described in Appendix A.

3.5. Operation component

The operation component searches for rules that involve previous items being combined in some way to form subsequent items. The rule that needs to be learned is the operation being used to perform the combination. In the case of Raven's matrices, the items are the visual features in the first two cells of a row, and the operations are different ways in which those visual features can be combined to form the third cell. Examples of different possible operations are given in Fig. 8.

The challenge in this component is that there are a number of potential operations that could be used to combine cells. We do not want to hand-build a separate system to look for each possible operation, we want one general operation-finding system. This can be accomplished by noticing that all the operations in the RPM can be reduced to the problem of detecting commonalities and differences in the previous cells. The operations amount to deciding whether the commonalities or the differences (or both) should be preserved in the subsequent cell. For example, the operation in Fig. 8(a) is equivalent to preserving the commonalities between the first two cells. Fig. 8(b) is an example of preserving differences. Addition—Fig. 8(c)—occurs when both the commonalities and differences are preserved. Thus detecting all the different operations has been reduced to the problem of detecting the commonalities and differences between two cells and deciding which of those sets are being preserved in the third cell.

When detecting commonalities, the set of components present in the first two cells is known, but not which of those vectors are in common. However, recall the superposition operation, which takes multiple vectors and combines them into a single vector that shares some similarity to each of its components. If the representations of the first two cells are superimposed, then any vectors that the two had in common will now be represented twice in the combined vector. This means that when comparing how similar the combined vector is to the components in the first two cells, any vectors that were in common should stick out as noticeably more similar. Thus it is possible to detect the commonalities between two cells by superimposing those cells, examining how similar the combined vector is to the candidate vectors, and identifying any candidate vectors that exceed a certain similarity threshold as common.

Differences can be detected in much the same way, except that in this case the representations of the two cells are combined through subtraction rather than addition. This means that any components the two had in common will be canceled out, and only the components that were different will be significantly similar to the component vectors (although some

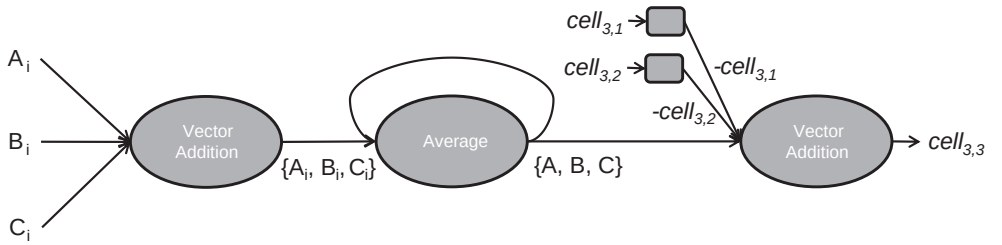

Fig. 7. Schematic diagram of the set component. The first population sums the three inputs (the three cells in a row) to create a set representation of the items in those cells. The second population averages those set representations across the rows. The final population subtracts the two cells in the third row from the average set representation, generating a prediction for the blank cell.

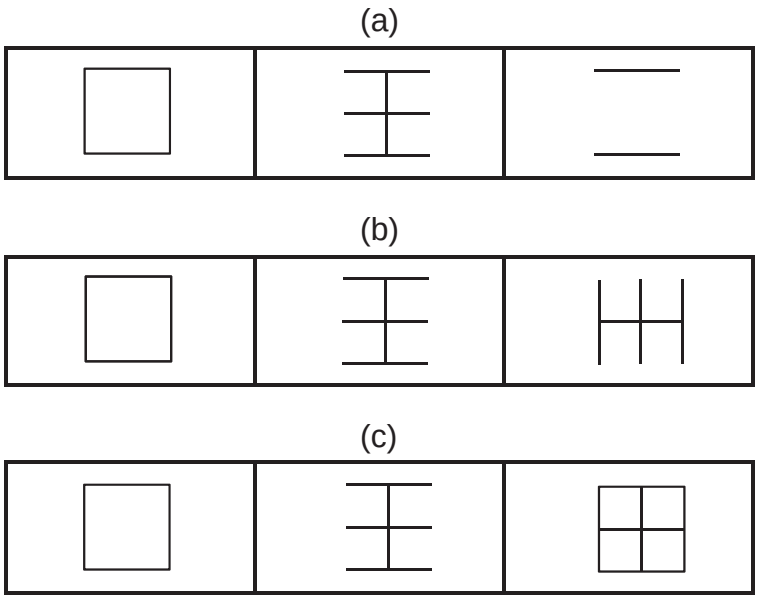

Fig. 8. Examples of different operation rules that could be used to combine two cells to form a third. (a) A "set intersection" type operation; if the first two cells are imagined overlayed on one another, only the overlapping components appear in the third cell. (b) An "exclusive or" type operation, where only the non-overlapping components appear in the third cell. (c) A "set union" type operation, where all components in the first two cells appear in the third.

may be negatively similar due to the subtraction). Thus the same method of combining, examining similarities, and selecting vectors that exceed a similarity threshold (positive or negative in this case) can be used to detect the differences between two cells.

Having detected the sets of vectors that are common and distinct between the first two cells, it is then possible to calculate the rule. The rule is the operation being used to combine the first two cells, or the relative influence of the common versus distinct set on the third cell. Since the third cell is known in the first two rows, we can determine that influence by calculating how similar the common/distinct sets are to the third cell. If the common set is similar to the third cell, that indicates that the operation is one which preserves common items. Doing the same for the set of differences enables the complete rule to be formulated, expressing how the two sets are combined to form the third cell. The answer is selected by applying the rule to the third row. The same network is used to detect the set of common and distinct features between $cell_{3,1}$ and $cell_{3,2}$, and then the rule is used to weight each of those sets. Summing the two weighted sets then gives a complete hypothesis for the blank cell.

The structure of this component is shown in Fig. 9. There are two parallel tracks: one for the items in common and one for the distinct items. The inputs, A_i and B_i, are the first two cells of the row. These inputs are passed through a network that calculates the set of common (X_i) or distinct (Y_i) features. The next population calculates the inner product (similarity) between the set vector and the third cell of the row, C_i. The "Average" population takes the average of those similarities across the rows. The resulting values, R_1 and R_2, represent the rule giving the weight that should be placed on the set of common and distinct features, respectively. Finally, the set of common and distinct features are weighted by those rule values and then summed to generate a prediction for the third cell, $\hat{C}i$. The neural implementation of this component is described in Appendix A.

3.6. Control structure

Each of the three neural components takes the Raven's matrix information as input and dynamically generates the rules that describe that matrix. That is the primary goal and purpose of this model. However, we would also like to be able to evaluate the success of those rules by running the model through the RPM, combining the output of the neural components to generate a complete rule set for each matrix (potentially involving multiple rule types), and giving the model a score indicating how many matrices were solved correctly. That is the purpose of the control structure. It is not implemented in neurons, but we have designed the controller to conform as closely as possible to the empirical data as well as the biological constraints imposed by the brain. It consists primarily of a simple system of rules that handles the input to and output from the neural components and controls when they run. Stewart et al. (2010) have demonstrated how such a control mechanism can be implemented using the NEF, and Eliasmith et al. (2012) showed how it can be combined with a simplified version of the reasoning system presented here. Given the significant increase in computational resources needed to implement and simulate the controller in neurons, we have not undertaken that step here; as discussed previously, in this work we seek to explore the more complex rule generation systems in depth, rather than multi-component integration. Consequently, building a neural control mechanism to go with the rule generation of this model remains for future work.

The primary reason a controller is required is that most Raven's problems involve multiple rule types (see Figs. 1 and

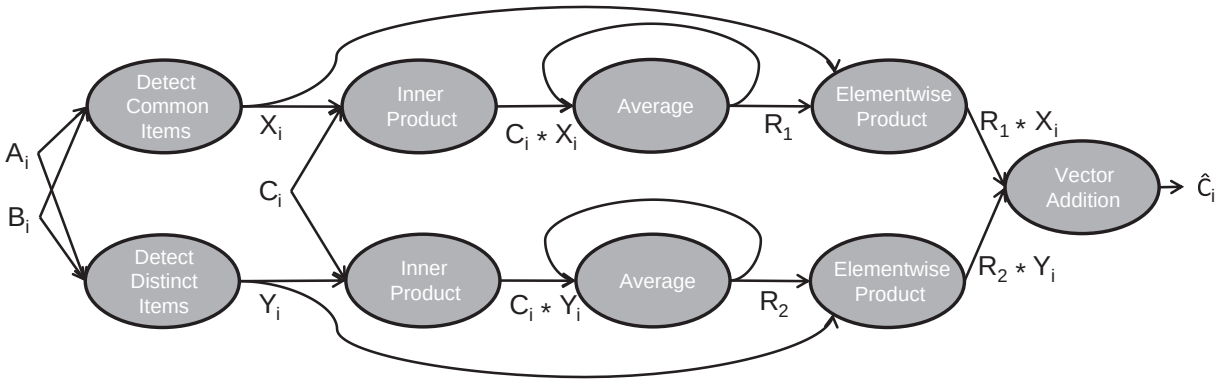

Fig. 9. Schematic diagram of the operation component. The two parallel tracks are for the common (top) and distinct (bottom) items. The first population in each track calculates the set of features that are the same or different, respectively, between the two inputs. The second population calculates the similarity between that set of features and the third cell in the row. The third population averages that similarity value across the rows. The fourth population multiplies the set of common or distinct items by the average similarity value. The fifth population sums the result from the two parallel tracks to generate a prediction for the blank cell.

3, which both involve sequence as well as set rules). Each neural component only understands the type of rule that it is built around; any other rules will appear as random noise. The components can tolerate significant noise and still find correct rules, but as more rules become involved this becomes increasingly difficult. Thus the controller's main job is to break the matrix down into subproblems involving fewer rules, run the neural components on those subproblems to generate all the rules for the matrix, and then use those rules to select an overall response.

The controller follows an iterative approach to carry out this task. It begins by giving each neural component the complete matrix as input. If none of the components are able to find a rule, it then breaks the problem down into simpler matrices and sends each of those to the neural components. It continues this process until either one of the components finds a rule or the matrix cannot be broken down any further. At the end of this process the components may have generated several rules, each describing one aspect of the overall pattern of the matrix.

In order to evaluate whether or not the neural components have found the correct rules, the controller tests whether those rules can be used to select the correct answer. To do this it uses the hypotheses for the blank cell that were returned by the neural components along with each rule. It checks how similar each of the eight possible answers is to a given hypothesis, indicating how well that answer matches what the model believes should be in the blank cell. Repeating this process for each generated hypothesis gives a total score for each answer indicating how well it matches the overall pattern generated by the model. This corresponds to a common answer selection strategy used by human subjects in the RPM, known as "response construction" (Vigneau, Caissie, & Bors, 2006).

A potential concern is that the real work of the model is being done by this non-neural controller, rather than the neural components described above. However, in the next section we will begin by showing that the neural components can function without a controller, accurately inducing rules and generating correct answers to matrices. The controller is only required to coordinate the neural components on problems involving multiple rule types. In addition, later in the results we will perform various manipulations to the model and examine the resulting impact on performance. All of these changes are applied only to the neural components—the controller is unchanged throughout. Thus the resulting performance changes can only be attributed to the processing of the neural components, and the corresponding discussion and conclusions pertain to those components rather than the coordinating role played by the controller.

4. Model experiments

There are two broad questions to be addressed with the results from this model, corresponding to the two axes identified in the Introduction section. First, does the model succeed in producing complex cognitive performance—can the model dynamically generate the rules needed to solve Raven's matrices? Second, of what value is the model's biologically detailed implementation—specifically, how can it be used to better understand the causes of cognitive decline in aging? In this section we will address each of these questions in turn.

For those interested in recreating these experiments, or running their own, a simplified, easy to run demonstration of the model can be downloaded from http://models.nengo.ca/RPMdemo. For the complete code, contact the authors. The model requires the Nengo simulation environment to run, which can be obtained (with installation instructions) at http://www.nengo.ca. Appendix A describes the typical parameter settings for the model in Table A1. Although there are many parameters involved in a complex biological model, we do not tune those parameters for each experiment. In almost all cases, the parameter values are the defaults listed in Table A1. In the discussion of the experiments we will always describe where the parameters differ from those defaults.

4.1. Demonstrating cognitive performance

4.1.1. Experiment 1. Finding a correct solution

In this experiment we will examine data from the neural components, without the controller, as they try to solve the

three matrices shown in Fig. 5. We generate one instance of each component and run it on the appropriate matrix; it is given a description of the matrix in VSA form as input, and tries to generate the rule describing the matrix and a prediction for the blank cell. As it runs we record the output from various stages throughout the component. This will help elucidate how the neural components generate rules, as well as demonstrate that they can find the correct answer using those rules.

We will begin with the sequence component. Fig. 10 shows the decoded signal from various stages throughout the component as it attempts to solve the matrix shown in Fig. 5(a). For a detailed explanation of how we decode a signal from neural spikes see Appendix A, but roughly speaking these signals can be thought of as the summed, weighted postsynaptic current induced by the neurons in a population, where the weights are vectors calculated to transform the current into the VSA domain. Note that this decoded output is only for the benefit of the reader; internally, the model is operating completely in neural spikes.

The signals being shown are the high-dimensional VSA vectors as they are represented in the neural populations, so the particular values are not meaningful to a surface viewing. What is of interest is the general transformation of information at each stage in the model. Fig. 10(a) shows the output of the population representing one of the inputs (one of the two adjacent cells for which a transformation will be calculated). In our model we present each cell for 200 ms. We take this to be a lower bound based on the duration of fixations revealed by eye-tracking data (Carpenter et al., 1990). However, this parameter is not critical to the model's performance, and the results are similar for presentation durations ranging from 100 to 900 ms. In Fig. 10(a) we can see that the signal—the VSA representation of that cell—switches every time the model examines a different cell. Fig. 10(b) displays the output of the network calculating the individual transformation between each pair of cells. Again we can see that the signal—the VSA representation of the transformation between the two cells—changes every time the cells change, but we can also note that there are some overall trends to the signal. This is because although the input cells are changing, the transformations between those cells all have something in common, which is the rule the component is trying to discover. Fig. 10(c) is the output of the averaging network, representing the average of all the individual transformations. The output of this network is the general rule for the matrix, capturing the trends observed in (b). Fig. 10(d) displays the convolution of that general rule with the second last cell of the matrix (i.e., the model's prediction for the contents of the blank cell, in VSA form).

Next we examine whether or not the sequence rule can be used to find the correct answer. We do this by examining how similar the component's hypothesis is to the eight possible answers. The component will calculate the inner product between its hypothesis and the vectors representing the eight answers; if it has generated a correct rule, then its hypothesis will be most similar to the correct answer. We set up a final population to perform this calculation, and recordings from this population are shown in Fig. 11. Fig. 11(a) shows the output spikes from this population, and Fig. 11(b) displays the decoded signal from those spikes. The eight lines represent the similarity between the hypothesis and the eight possible answers; the similarity to the correct answer is shown in black. At the end of the run, once the model has received all of its input, it selects the most similar answer as its final choice. We can see that over time, as the model generates a more accurate rule, the hypothesis for the blank cell becomes increasingly similar to the correct answer. This demonstrates that the component was able to find the appropriate rule for this matrix.

We can do the same analysis for the set and operation components. For brevity's sake we will only show the comparison between the component's hypothesis and the eight possible answers. The results of the set component applied to Fig. 5(b) and the operation component applied to Fig. 5(c) are shown in Fig. 12. It can be seen that each component can find the correct answer for rules of its type. In the next experiment we examine the performance of the complete model combining all three components.

4.1.2. Experiment 2. Matching human performance

We now address the question of how the model performs on the Raven's Advanced Progressive Matrices test, and how it compares to human performance. In this analysis we generate a population of 30 models (as opposed to the single instances in Experiment 1), and run each model through the 36 items of the RPM. Variation is created in the population due to the stochastic nature of our methods: the randomness inherent in VSAs, realistic, distributed neural properties, and heterogeneous neuron tuning (see Appendix A). We can then examine how many problems, on average, the models solve correctly. We do not expect the models to solve the RPM perfectly; in fact, perfect performance would raise doubts as to the realism of the model, since such performance in human subjects is extremely rare. In first year undergraduate students, average performance is around 22 problems ($\sigma = 5.6$) solved correctly (Bors & Stokes, 1998). After running the experiment, the result for our model is an average score of 18.4 problems solved correctly ($\sigma = 2.4$). As with human performance, the successes and failures are distributed throughout the 36 problems of the RPM, but the model tends to perform better on earlier problems than later (the correlation between average performance and problem number is $r = -0.41$).

In this simulation the model is performing slightly worse than the human subjects. However, we can show that this discrepancy is a product of computational constraints, not theoretical ones. There are two main factors that determine the accuracy of the model. The first is the dimension of the VSA vectors used; the higher the dimension of the vectors, the more accurate the VSA computations (Plate, 2003). The second is the number of neurons used to represent those vectors; the more neurons in the model, the more accurately it can implement the vector level computations (Eliasmith & Anderson, 2003). Unfortunately, increasing either of those factors increases the computational demands of the model. Current computing power forces us to set their levels much lower than we would expect in the human brain. Based on neurophysiological data such as projection ratios, cell density, and firing rates in the mammalian cortex, and psychological data such as vocabulary sizes, structure depth, and accuracy in human subjects, Eliasmith (2013) estimates that humans could employ approximately 500 dimensional vectors, with about 35,000 neurons

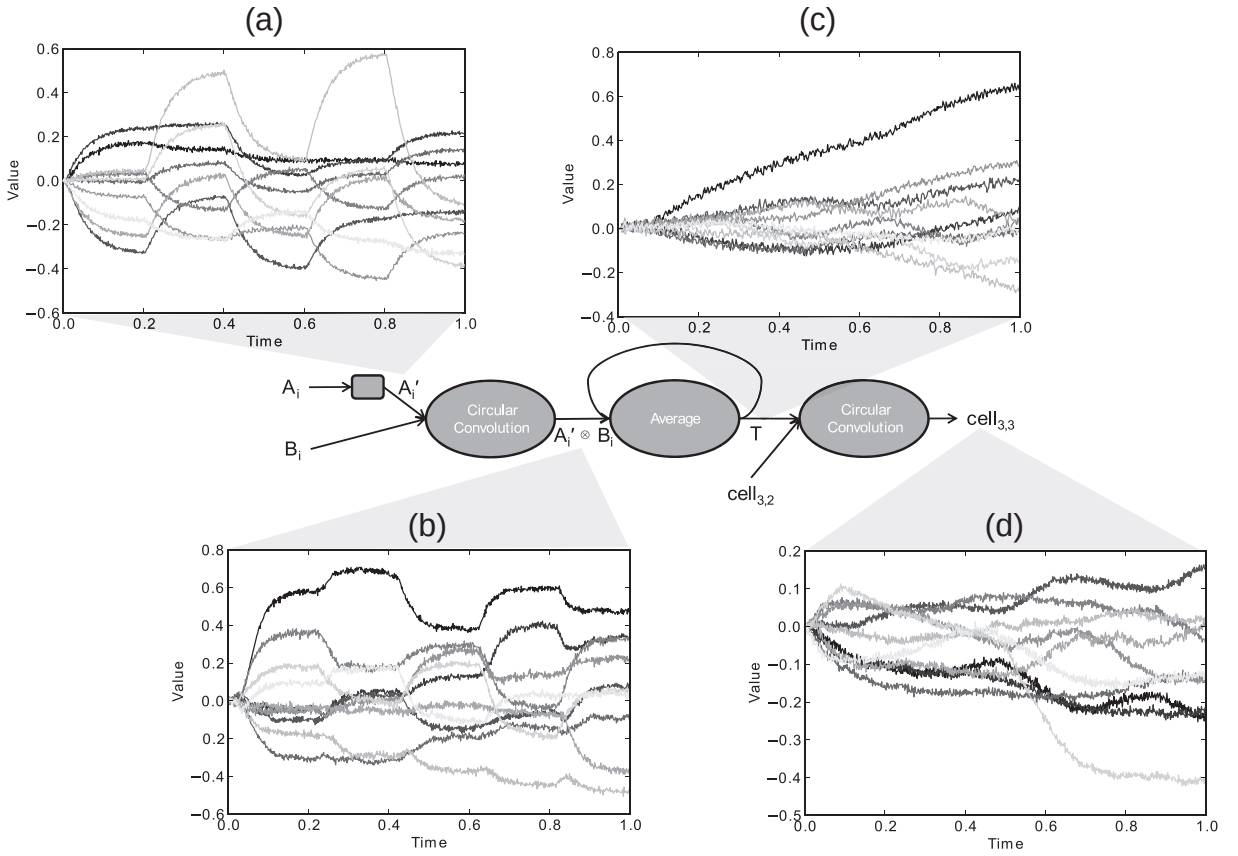

Fig. 10. Decoded signals from various stages throughout the sequence component. All of these signals are high-dimensional VSA vectors being represented in neural populations; each line represents the value of one dimension of the vector (we are only displaying 10 of the dimensions in order to reduce clutter). (a) One of the inputs, the first of the two adjacent cells for which an individual transformation will be calculated. (b) The individual transformation between the two cells.(c) The output of the network calculating the (approximate) average of those individual transformations over time (the general rule for the matrix). (d) The hypothesis of the model as to the contents of the blank cell.

per vector (i.e., 70 neurons per dimension). In our model we are limited to 30 dimensional vectors, with about 900 neurons per vector (30 neurons per dimension).

We can take some computational shortcuts, sacrificing neural realism for faster speed, in order to investigate how the model would perform given more realistic numbers of dimensions/neurons. Our model is set up so that it can be simulated using only the population-level dynamics, rather than individual neurons. The calculations carried out in the model are the same, but the vector-level computations are being simulated directly rather than simulating the neurons carrying out those computations. This dramatically reduces the computational load of the model, as tens of thousands of neurons no longer need to be simulated, and these savings allow us to increase the vectors to 500 dimensions. If we perform the same analysis—generating a population of individuals and running them through the 36 problems of the RPM—average performance is 22.3 ($\sigma = 1.2$) problems solved correctly, within the range we find in human subjects. This suggests that, given sufficient computational power, the theory of dynamic rule generation we have developed can be used to find rules that correctly solve Raven's matrices at least as well as human subjects. We should emphasize that the claim here is not that the model's performance is equivalent to humans' (for example, human performance typically has much higher variance than the results shown here); rather, we seek to show that the model's performance falls within an acceptable range established by empirical data.

Note that all results in subsequent experiments involve the full model in which all individual neurons are being simulated, not the simplified version used to explore performance with 500 dimensional vectors.

4.1.3. Experiment 3. Error analysis

In addition to comparing to human correctness, we can ask whether or not the model makes the same kinds of errors as humans when it gets the answer wrong. Useful data in this regard comes from Babcock (2002). In her study she classified the 7 incorrect answers in each matrix of the RPM according to what type of reasoning failure would cause a subject to select each response. Her categories, based on the earlier work of Forbes (1964), were *a*) incomplete correlate (IC), in which "the individual does not identify all of the relevant variables needed to determine the correct figure", *b*) wrong principle (WP), in which "the person's answer reflects reasoning that is qualitatively different from that which results in a correct solution to the

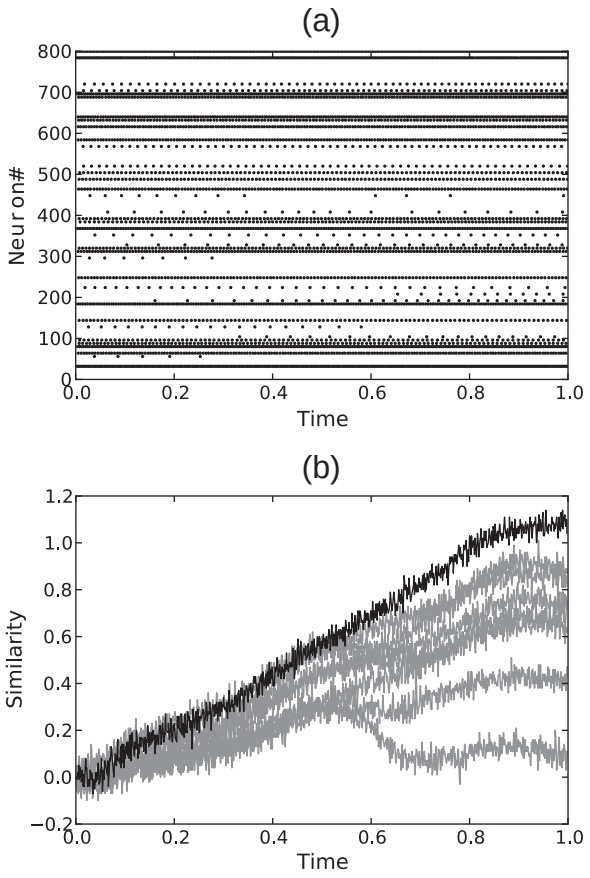

Fig. 11. Recordings from the population calculating the similarity between the model's hypothesis for the blank cell and the eight possible answers. (a) A spike raster—each row is one neuron in the population, and each dot is a spike from that neuron. We are displaying only 10% of the neurons in order to make the spikes more visible. (b) The decoded signal from those spikes. The eight lines represent the similarity to the eight possible answers. The correct answer is shown in black, indicating that the model has correctly calculated that answer to be the best candidate for the blank cell.

problem", *c*) confluence of ideas (CF), in which "the person does not understand that some of the elements in the problem do not change and are not relevant to the solution", and *d*) repetitions (RP), in which "the individual simply selects a figure that is adjacent to the empty cell of the problem". She then ran 818 subjects through the Advanced Progressive Matrices and analyzed how often subjects made each type of mistake.

We can perform the same analysis on our model's performance. Using the same classifications as Babcock, we generated a sample population of 30 models, ran them through the RPM test, and analyzed how often the models made each type of error. We make some adjustments to our standard procedure in order to better reflect the Babcock study. As Babcock describes, subjects "were allowed 20 min [40 min is the standard time limit] to solve as many of the problems on the Raven's APM as possible. Although they were encouraged to attempt the problems in the order they were presented, subjects were allowed to omit responses to problems without penalty in their overall score. The result of the time limit and the instructions to the test is that not all subjects responded to all of the problems...." The standard behavior of our model is to not select a response if it considers two or more choices to be equally likely, and usually we mark a "non-choice" as incorrect. But to match the Babcock analysis, in this case any problems where the model did not make a choice were not penalized, and omitted from the error analysis. In addition, because of the time restriction, Babcock had very few subjects attempt the last 5 matrices, and so only performed the error analysis on the first 31 problems. We therefore similarly restrict our analysis to those matrices.

Babcock (2002) divided her data according to the ability level of the subjects (based on their Raven's score). With an average score of 18.4, our model falls between her medium ability group (11–17) and high ability group (18–31). In Fig. 13 we show the error analysis of our model population versus those two groups of human subjects. It can be observed that in general the frequency with which the model makes each type of error is in line with what we would expect given the model's overall performance, falling in between the two groups. The only exception to this is in regard to the repetition (RP) errors, which our model seems to engage in more frequently than human subjects. The cause of this divergence is uncertain; one possible explanation is that humans have some a priori heuristic that suggests that problem solutions are unlikely to be simple repetitions of the question, which is lacking in the model. It is also possible that this mismatch is not significant, and the model falls within the expected range of human data—we cannot be sure without the standard deviations for the human subjects. However, we can see that in general the model's error data is in line with the human data. This provides additional evidence that not only does the model solve the matrices with similar performance to human subjects, when it gets the answer wrong it does so in a similar fashion to human subjects.

4.1.4. Experiment 4. Examining rule generality

Next, we examine whether or not these vectors returned by the model are truly rules. That is, it is unclear when the model returns a rule if the model has found regularities appropriate just to the current matrix, or if it has truly induced general rules that could be broadly applied. For example, there are two ways a young child might be able to generate the answer "4" to the question "what is 2 + 2?". First, she could simply memorize that when she sees the pattern 2 + 2 the answer is 4. Second, she could have learned the general rule of addition, which can be applied to add any two numbers. We can differentiate the two by examining the ability of the child to generalize; can she use the same method to solve other addition problems? If her accuracy is much higher on problems she has already seen, that suggests she is employing the former technique; if her accuracy is as high on new problems as it was on old problems, that suggests the latter. That is the test we now apply to the rules generated by the model.

For this purpose we created a mini-RPM test with 9 problems. The first three problems are normal RPM-style problems, one for each rule type (the three problems shown in Fig. 5). We then created two alternate matrices for each of those original matrices. The alternate matrices had the same underlying rules, but different implementations of those

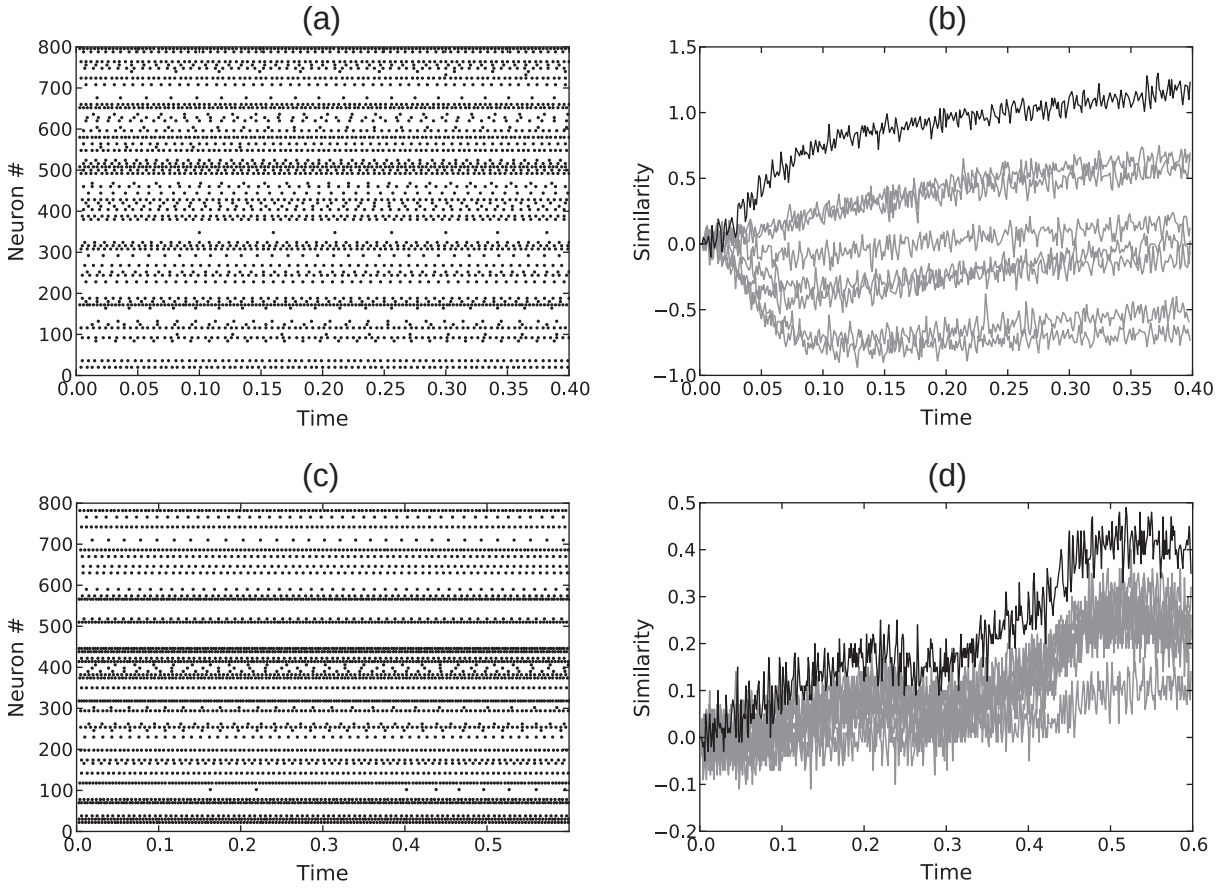

Fig. 12. Recordings from the population calculating the similarity between the model's hypothesis for the blank cell and the eight possible answers for the set (a,b) and operation (c,d) component.

rules (see Fig. 14). To test the model we ran it on the first three matrices, allowing it to generate a set of rules for each matrix. Then in order to test the generality of those rules, we applied them, without any modification, to the two alternate matrices. We are taking rules that were generated for one matrix, and trying to use them to solve a matrix with the same underlying rules but different values—analogous to using the rule for solving $2 + 2$ to solve $4 + 5$. If the rules are truly general, then they should work as well on the alternate matrices as they do on the original matrices.

We generated a population of 30 models and ran them through this procedure. Fig. 15 shows the difference, for each component, between the proportion of the population that correctly solved the original matrix (for which the rules were generated) and the population's score when using the same rules on different matrices. It can be seen that performance is almost identical in the two cases. The largest change is actually an increase of 5% for the operation component (i.e., 5% more of the population solved the novel matrices correctly than the original matrix). The confidence intervals indicate that none of the changes are statistically different from zero. These results reveal that the rules generated by the model truly are general; they capture the abstract relationships of the data, so they work equally well no matter how those relationships happen to be instantiated in a given problem.

4.1.5. Experiment 5. Performance across tasks

Another way to illustrate the generality of the system is to apply the same model to multiple tasks. That is, if this model contains generic rule finding abilities, then it should be able to generate rules on tasks other than the RPM. For this demonstration we chose the figural analogy problems from Evans (1968), seminal work on the computational modeling of analogy.[6] We chose this task because, like the RPM, it does not rely on linguistic abilities or background knowledge; the problems are all expressed through transformations of geometric shapes. However, rather than a 3×3 matrix, the Evans task consists of 20 "a is to b as c is to?" analogy problems. The Evans problems also require different specific rules to solve, although we will show that they can be captured by the rule generation systems of our model.

The advantage of the similarity between the two tasks is that we can build a single model and apply it to both tasks. We use the same vocabulary to describe the two tasks, the same model, and all the same parameter settings. All that changes is the input to the system, as instead of having a 3×3 matrix and eight possible answers, the model sees the *a*, *b*, and *c* figures and five answers to choose from.

[6] Lovett and Forbus (2012) have also applied their modelling techniques to the Evans problems.

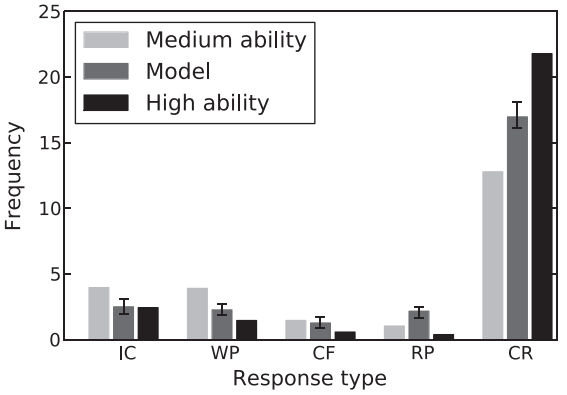

Fig. 13. Analysis of error types, comparing performance of medium ability human subjects, high ability human subjects, and the model. IC = Incomplete correlate, WP = Wrong principle, CF = Confluence of ideas, RP = Repetitions, CR = Correct response (see text for details).

To explore the model's performance across tasks we created a sample population of 300 "individuals"—models with randomly varying number of neurons (ranging between 10 and 30 neurons per dimension) and vector dimension (ranging between 30 and 60). We then ran each model on both the RPM and Evans' analogy problems. Fig. 16 shows the distribution of performance across the two tasks. The first observation to be made is that the model is able to successfully solve Evans' task, achieving performance up to 18/20 problems correct. This reveals the power and generality of the rule generation abilities of this model. It was not necessary to add new rules or new abilities to the model to solve the new task, the system was able to generate the appropriate rules based only on the new information in the problems.

Since we are using the same model for both tasks, we can compare the model's performance across tasks. This allows us to begin to examine the fundamental principle of general intelligence, namely, that individuals that perform well on one task tend to perform well on other tasks. In general, Fig. 16 shows the trend we expect of positively correlated performance. We are not aware of any data specifically comparing performance on the RPM and Evans' problems; however, Evans' problems are drawn from a larger test, the quantitative section of the American Council on Education Psychological Examination for College Freshmen (ACE-Q), for which data is available. Bolin (1955) reports a correlation between the RSPM and ACE-Q of $r = 0.59$, and the correlation obtained from the data in Fig. 16 is $r = 0.64$ (95% confidence intervals 0.56–0.71).

This is only a basic demonstration of cross-task performance, on two tasks that share a fairly similar structure. It would be interesting in the future to explore performance across a range of more diverse tasks, to begin to examine the processes underlying the positive manifold of *g*. However, this experiment serves as a demonstration of the generality of the rule generation abilities of this model, and hints at the value that generality can provide.

4.2. Investigating cognitive decline

Having addressed the cognitive performance of the model, we can now demonstrate the advantages of its biologically detailed implementation. For this we have chosen to examine how the model can be used to explore the link between neurophysiological changes and behavioral performance in cognitive decline. There is a large amount of behavioral evidence concerning how performance on these types of complex cognitive tasks is affected by age (Deary, Whalley, Lemmon, Crawford, & Starr, 2000; Der, Allerhand, Starr, Hofer, & Deary, 2010; Kaufman et al., 1989; Morse, 1993; Salthouse, 2009; Tucker-Drob, 2010). There is also good evidence concerning how brains change with age (Fjell et al., 2006; Pakkenberg & Gundersen, 1997; Park et al., 2010; Raz et al., 2005; Resnick, Pham, Kraut, Zonderman, & Davatzikos, 2003; Staff, Murray, Deary, & Whalley, 2006; Zarahn, Rakitin, Abela, Flynn, & Stern, 2007). What is less clear is the link between the two: how are the changes in the brain connected to the changes in performance? This is much harder to establish, due to the large number of aging factors and complex interactions between those factors. We know that the brain changes are linked to the performance changes, but it is difficult to establish this empirically because we cannot manipulate the aging factors in a controlled way. See Deary et al. (2009) and Salthouse (2011) for detailed discussions of this issue and the available evidence. That is the gap into which we place this model: we will examine how the model's performance is affected when manipulating the model in ways that recreate the effects of aging, providing evidence for the quantitative impact of the neurophysiological changes on performance. This is made possible by the combination of complex cognitive performance and biologically detailed implementation.

We investigate two of the leading theories of neurophysiological change associated with aging: neuron loss and representational dedifferentiation. We chose these two factors in particular because, as we will see, they can be mapped onto manipulations at the different levels, VSA and NEF, of representation in the model. Thus these factors are a good demonstration of the different ways in which the model can be used to explore cognitive decline, as well as the model's ability to integrate manipulations at different levels of abstraction.

4.2.1. Experiment 6. Neuron loss

The first aging factor we will examine is neuron loss. The total number of neurons in the brain increases until about the age of 20, and then from the age of 20 to 90 decreases by about 10%—around 85,000 neurons per day (Pakkenberg & Gundersen, 1997). Given that neurons are the primary processing component of the brain, it is natural to suppose that having fewer of them would be detrimental to performance. This can be seen empirically in correlations between brain/gray matter volume (an indirect approximation of neuron number) and performance (Narr et al., 2007; Royle et al., 2013). However, gray matter is a very rough measure of neuron number, and more importantly we would like to make causative rather than correlational claims. In other words, we would like to see that when we change the number of neurons, performance changes in response.

This is a question that can be directly studied in our model; as it is implemented in spiking neurons, we can manipulate the number of those neurons and examine how that impacts the model's performance. We generated an initial population of 60 models with 15 neurons per

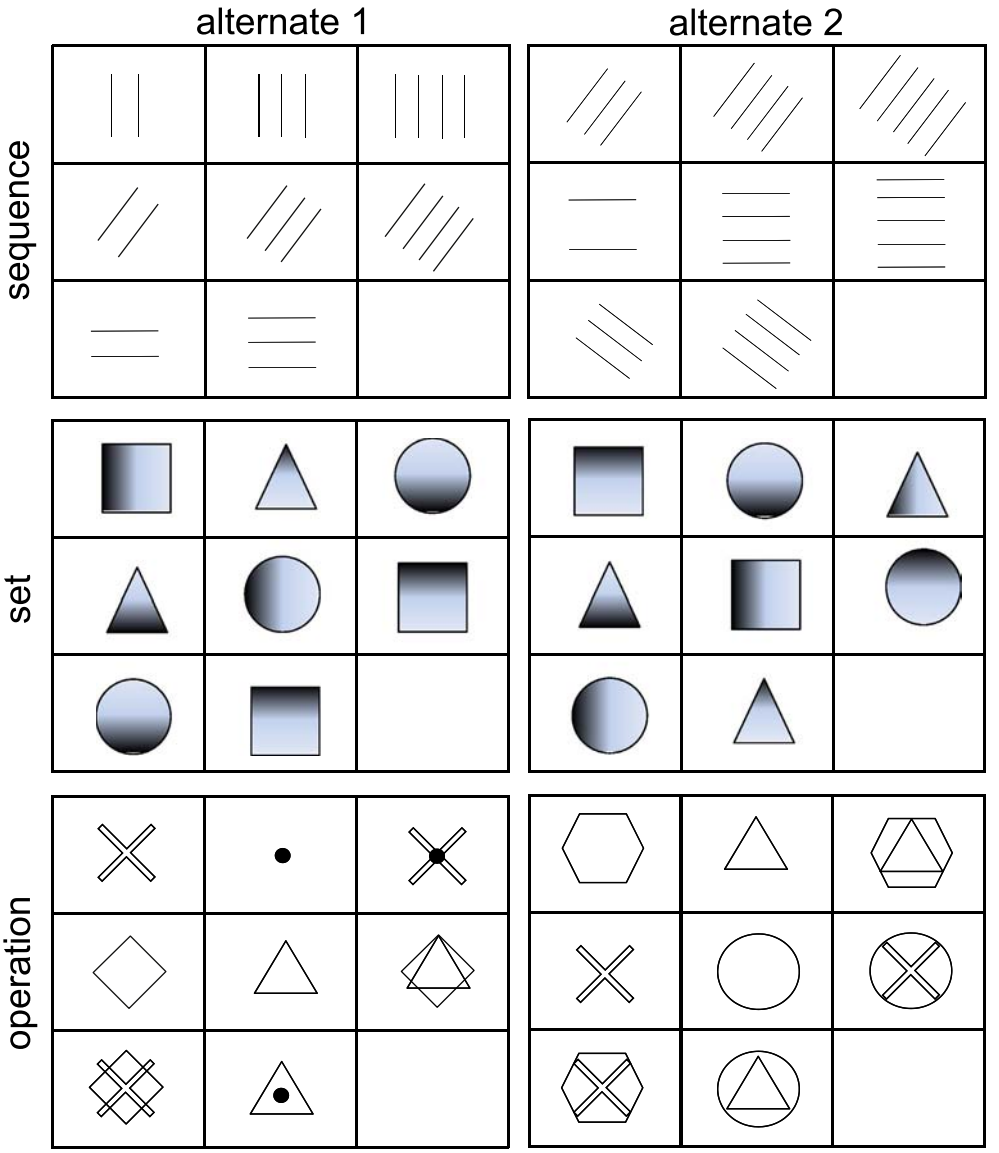

Fig. 14. Alternate matrices used in the rule generalization experiment (Experiment 4), containing the same underlying rules as in Fig. 5 but different implementations.

dimension, ran them through the 36 items in the Raven's test, and recorded their overall scores. We then generated populations with decreasing amounts of neurons, and repeated the process.[7] The effect of this manipulation on the model's average performance is shown in Fig. 17. In this case we have ranged between 0 and 20% neuron loss, although the Pakkenberg and Gundersen (1997) study gave an average of 10% neuron loss throughout a lifetime. This is because the 10% figure is for the whole brain, whereas several studies have shown that the frontal/parietal areas most often associated with RPM-type reasoning are particularly affected by neuron loss in aging (Raz et al., 2005). The latter studies do not give specific figures for neuron loss, so we take 20% as a rough upper bound.

There are a number of observations to be made from this figure. First, as the number of neurons decreases, so too does the performance of the model ($r = -0.31$). Second, the performance trend shows a number of "human-like" characteristics not captured in previous models: a range of performance at each point, and graceful degradation across the manipulation. The former is a result of the stochasticity of VSAs and biologically detailed NEF modeling, and the latter is made possible by the distributed (at both the vector and neural level) nature of our representations, allowing the system to model subtle changes rather than adding/removing entire symbolic components. Third, the performance loss is

[7] We decrease the number of neurons equally throughout the entire model, as given the relatively small area into which our mapping places the neural components of the model (Section 3.2) we do not have sufficient resolution from neurophysiological aging studies to justify a more fine-grained distinction.

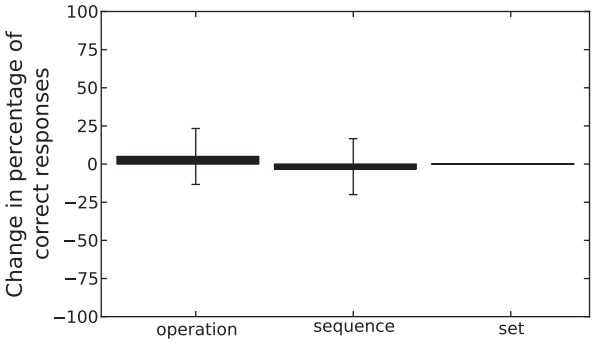

Fig. 15. Testing the generality of the generated rules by applying them to novel matrices. Displaying the difference between the proportion of the population that gave the correct answer on the matrices for which the rules were generated and the population's score when applying the same rules to novel matrices, with 95% confidence intervals. These results demonstrate that the model learns rules that generalize to novel matrices.

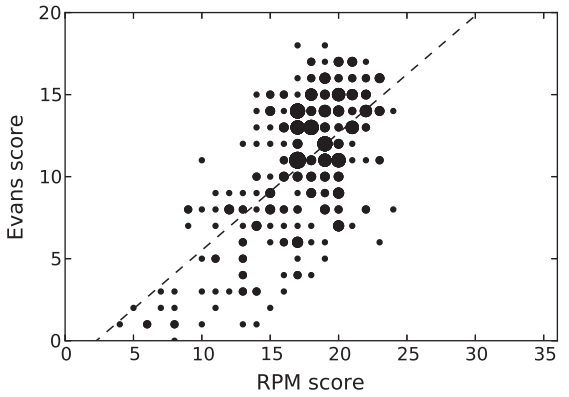

Fig. 16. Scatter plot of individual models' scores on the RPM and Evans' analogy problems. The size of the circles indicates the number of individuals falling into each data point, since many of them overlap, and the dashed line indicates the best linear fit to the data.

less than that observed in aging human subjects (typically around 8 fewer matrices solved correctly between young and old age — Babcock, 2002; Salthouse, 1993). One explanation of this difference is the point mentioned above: effects measured in human subjects are a conglomeration of many neurophysiological changes (neuron loss, demyelination, connectivity loss, etc.), whereas in this experiment modeling has allowed us to isolate the effects of a single aging factor. This is interesting, as it suggests that the model, and perhaps the human brain, can tolerate significant amounts of neuron loss (in isolation) without dramatic performance declines.[8] After exploring the next aging factor we will examine how the factors combine to impact performance, as they do in human subjects.

We can also examine how the model's performance changes with respect to the three different rule types. We find that performance on matrices involving a set rule drops the most (16% decrease in performance), then operation matrices (11% decrease), and then sequence matrices (5% decrease). This suggests that the set component is most sensitive to neuron loss, followed by the operation component and the sequence component. However, this is complicated by the fact that the performance of the components is interrelated, especially because many matrices involve multiple rule types. A complete analysis of this issue would require a much more thorough investigation than we seek to undertake here, but this result presents an interesting suggestion of other potential explorations made possible by this model.

4.2.2. Experiment 7. Representational dedifferentiation

The second aging factor we investigate is the neural dedifferentiation hypothesis. This is the idea that older brains do not represent different concepts as distinctly as younger brains. Studies providing evidence for this effect involve scanning subjects using fMRI while they observe different categories of images (Park et al., 2004, 2010) or hold different concepts in working memory (Payer et al., 2006; Zarahn et al., 2007). This makes it possible to estimate how "distinctly" subjects are representing concepts by measuring the degree to which different concepts result in different neural activation patterns. In all cases the finding is that distinctness is negatively correlated with age; as subjects get older, they begin to blend their representation of different concepts. In addition, Park et al. (2010) find this dedifferentiation to be particularly correlated with the age-related decline in fluid processing ability, the primary ability associated with tasks such as the RPM.

We can apply this result to our model by manipulating the dimension of the VSA vectors. Recall that in Fig. 2 we showed that in order to represent more vectors in a vocabulary while maintaining an upper bound on the similarity between those vectors, we need to increase the dimension of the vectors in the vocabulary. This is equivalent to saying that in order to make the vectors in the vocabulary more distinct, we need to increase their dimension. Thus the aging process, in which neural representations of different concepts become less distinct, can be modeled by decreasing the dimension of the VSA vectors. Note that neuron loss will also have the effect of decreasing the accuracy of representation (among many other effects), thus the two manipulations are related. However, the question being addressed here is whether changes applied only at the level of vector representation, independent of any other changes, can account for aging effects. We measure the "differentiability" of a vocabulary of vectors by calculating the average proportion of vectors that

[8] However, note that the losses associated with aging are gradual, and so the brain has time to adjust and compensate for these losses as best it can (Fjell et al., 2006). This is represented in the model by recalculating the synaptic connection weights between populations after decreasing the number of neurons. If we do not include this compensation mechanism, then performance loss is much more severe (around 75–90%). This could be likened to measuring RPM performance immediately after sudden brain injury, such as from trauma or stroke, delivered to the whole Raven's reasoning mechanism. That is not the focus of this work and so we make no strong claims on these results, they are simply intended to illustrate the effect of the compensation.

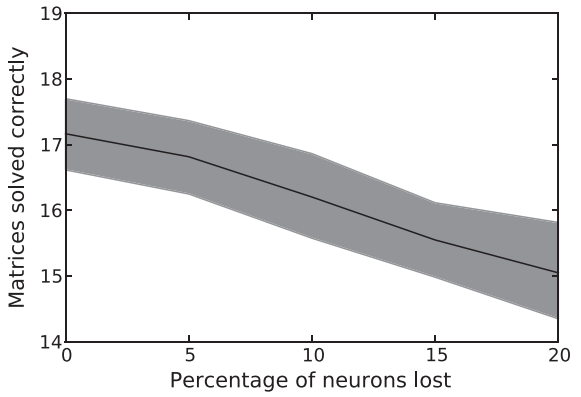

Fig. 17. Result of decreasing number of neurons on the model's overall performance (the number of Raven's matrices solved correctly out of a possible 36). Displaying 95% confidence intervals.

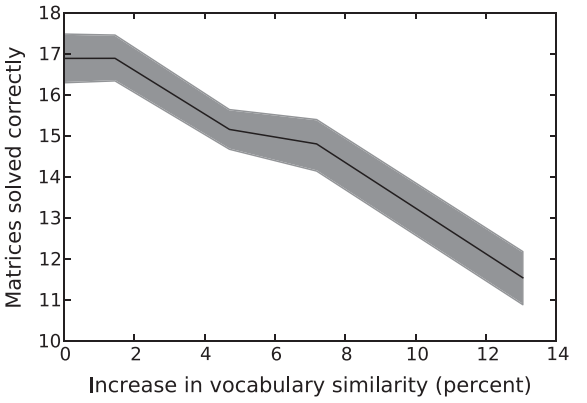

Fig. 18. Result of increasing vocabulary vector similarity on performance, with 95% confidence intervals. Vocabulary similarity is measured as the average percentage of the vocabulary that exceeds a fixed similarity threshold to any given vocabulary item.

are more than 30% similar to each vector in the vocabulary (the particular percentage is not important, it simply establishes a consistent threshold). This can be thought of as the average proportion of the vocabulary that could be confused with any particular vector. Thus as we decrease the dimension of the vectors, the proportion of confusable vectors increases.

Again we simulate this process by generating an initial population of 60 models (with 30 dimensional vocabulary vectors), analyzing their performance, and then repeating the process in populations with decreased vocabulary dimension. The result of this manipulation is shown in Fig. 18, where it can be seen that decreasing distinctness of representations leads to decreasing performance ($r = -0.59$). Note that unlike with neuron loss we do not have quantitative measures of the degree to which specificity changes with aging, and so we cannot be as exact about the range of the manipulation as we were in Experiment 6. However, what we can say is that dedifferentiating neural representations directly results in declining performance.

The next question we can ask is how these factors interact; when subjects both lose neurons *and* begin representing concepts less distinctly, as likely happens in true aging brains, how does that combination impact performance? For this we carried out a 2×2 ANOVA with neuron number and vector dimension at pre- and post-aging levels (post-aging defined as the maximum range of the manipulations applied in the previous experiments). The result was that although both neuron number and vector dimension were significant ($p < 0.05$), their interaction was not. This suggests (although not conclusively) that these two aging factors combine in a simple additive fashion; this could be due to the related nature of the neuron loss and dedifferentiation manipulations, as mentioned earlier, or indicate that they represent two distinct and independent mechanisms. We are not aware of any studies investigating the combined effect of neuron loss and representational dedifferentiation, and so this represents a prediction from the model. This analysis demonstrates an advantage of the modeling methods we have employed: we are able to combine the manipulations from the two levels of representation—neural and VSA—in the performance of one model, enabling us to examine the multiple levels of change present in human subjects.

4.2.3. Experiment 8. Aging error patterns

We can also examine how the model's error patterns change with age. For this we return to the analysis from Experiment 3, using the data of Babcock (2002). Babcock analyzed the error patterns of her subjects divided into young (18–30), middle-aged (31–59), and elderly (69–90) groups. Thus we know how error patterns on the RPM change in human subjects, and we can compare that to how the model's performance changes under the two aging manipulations.

Just as in Experiment 3, we created a sample population of 30 models, ran them through the RPM, and analyzed which errors they made. As an approximation of the young, middle-aged, and elderly groups used by Babcock, we created populations at the lower, middle, and upper end of the manipulations described in Experiments 6 and 7. The comparisons between the human and model data are shown in Fig. 19. It can be seen that the model follows the general trends of the aging humans. As we would expect based on the results of Experiments 6 and 7, the magnitude of the changes are smaller under the neuron loss manipulation. Unfortunately we cannot statistically compare the fit of these different manipulations to the human data, as the variance of the human data is not available, so it is difficult to make strong claims based on these results. However, this experiment serves to illustrate the value of combining complex cognition and biological detail, which allows us to link low level neurophysiological changes to high level variables such as error patterns.

4.2.4. Experiment 9. Individual differences in aging

In this experiment we look at how this model can address individual differences and their trends in cognitive decline. Longitudinal and cross sectional studies have provided a number of results on how performance changes in an aging population. Mean population performance decreases (Salthouse, 2009), which is unsurprising given that individual performance tends to decrease. More interestingly, aging populations also show an increase in the variance of performance (Morse, 1993). Finally, a

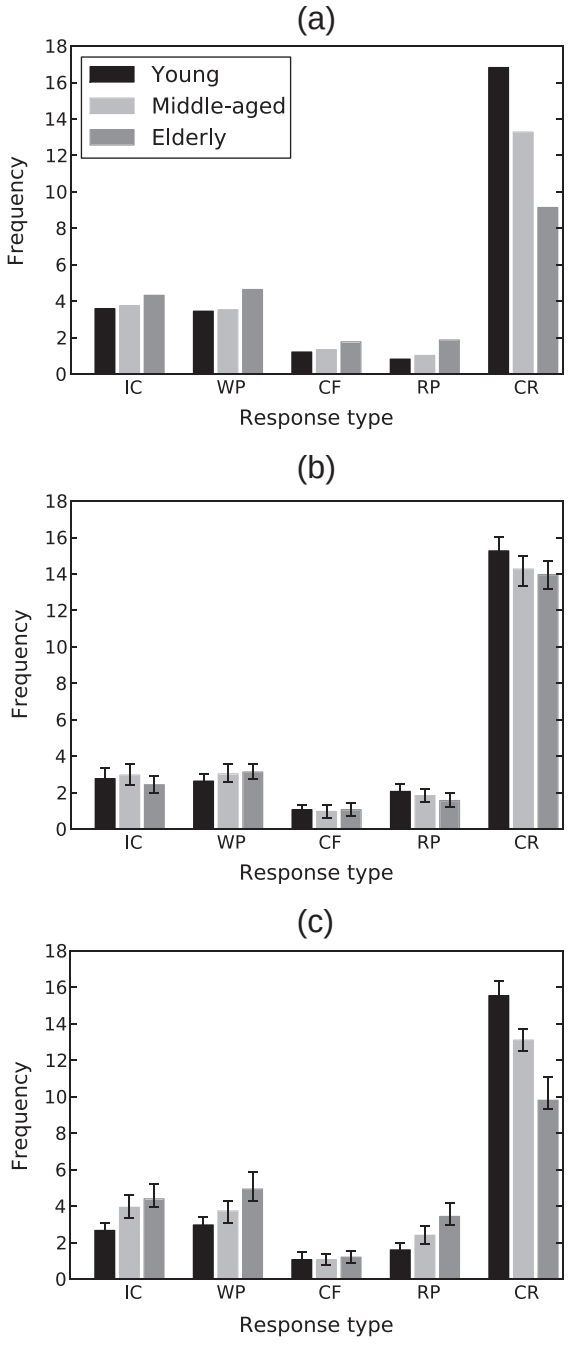

Fig. 19. Comparison between (a) aging human error patterns, (b) model errors when manipulating neuron number, and (c) model errors when manipulating representational distinctness. Displaying 95% confidence intervals for the model data. IC = Incomplete correlate, WP = Wrong principle, CF = Confluence of ideas, RP = Repetitions, CR = Correct response (see Experiment 3 for details).

consistent result is that despite these decreases in average performance and increases in variance, relative performance of individuals in the population is fairly well preserved (Deary et al., 2000); in other words, a subject first in his cohort at age 20 will likely still be at (or near) the top of his cohort at age 80. The

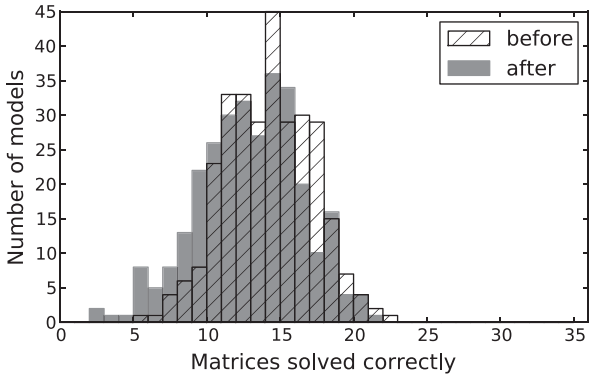

Fig. 20. Histogram of population performance before and after aging. The aged population shows a downward shift in performance, as well as greater variability.

question we want to answer is whether or not a set of models displaying individual differences will follow these same trends when we simulate the aging process.

For this analysis we generate a sample population of 300 individuals. These individuals are generated by creating models with randomly varying number of neurons (ranging between 10 and 30 neurons per dimension) and vector dimension (ranging between 8 and 22), establishing a range of initial ability. This is consistent with empirical work demonstrating that individuals vary in neuron number and representational specificity, and that those differences correlate with reasoning ability (Narr et al., 2007; Park et al., 2010). Note that we do not mean to suggest that such a simplistic manipulation completely accounts for the complex individual differences observed in humans. Rather, we want to show that these factors can create a range of individual ability, and examine how those individual abilities change with age. For an example of a different exploration of individual differences based on working memory, see Appendix C, or for the impact of changes to the control structure see Rasmussen and Eliasmith (2011).

We next age the entire population by decreasing the number of neurons in each individual by 20%. We have chosen to manipulate only the number of neurons in the aging process, rather than including vector dimension, because as mentioned earlier we do not have good data on the degree to which representational specificity changes with age. This makes it difficult to manipulate the two variables in parallel, because we do not know what degree of specificity change is appropriate for a given loss of neurons. To keep the results as clean as possible, we manipulate only the variable for which we have good data in this analysis.[9]

The pre- and post-aging performance distribution is shown in Fig. 20. Note the characteristic normal distribution of performance in the initial population, and its downward shift and "flattening" after aging. In Fig. 21 we have plotted both the mean and coefficient of variation (standard deviation divided by the mean) of the population relative to the proportion of neurons lost. This demonstrates exactly the

[9] We did investigate the effect of changing vector dimension, and the results differed in magnitude but had the same trends as those reported here.

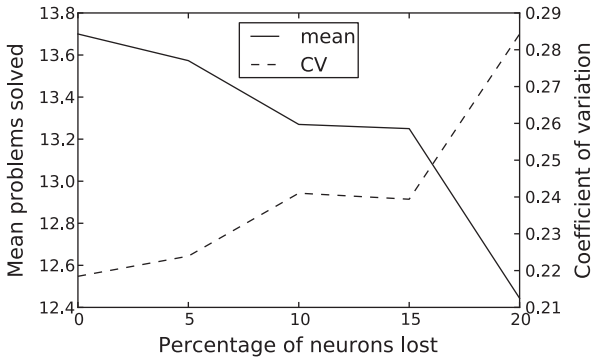

Fig. 21. Mean and coefficient of variation of population performance with aging. Aging corresponds to decreasing average performance and increasing variation, consistent with data from human populations.

trends described in the empirical research: decreasing average performance and increasing variance. Next we can examine the performance ranking of the individuals in the pre- and post-aging population. We find that the two rankings are moderately correlated, Spearman's $\rho = 0.61$ ($r = 0.58$ for the raw scores). This is consistent with human data, which shows correlations between the performance rankings at young and old age ranging between approximately 0.5 and 0.8 (Deary et al., 2000). As with all of the analyses in this section, these behavioral results are not new; it was already known that as a population ages we see performance decreases, variability increases, and relative ranking remaining consistent. What is new is the connection between neurophysiological changes and performance changes—we have shown that those known behavioral results can be directly explained by changes in the underlying neurophysiological variables.

5. Discussion

We began by arguing for the importance of models that combine complex cognitive performance with biological detail. The previous section presented a broad range of results, in order to investigate both of these issues. Here we will summarize those results, as well as discuss what they reveal concerning the advantages and disadvantages of both the model presented here and this style of modeling in general.

With respect to the cognitive performance of the model, we have shown that the model is able to dynamically generate the rules that describe a Raven's matrix using networks of spiking neurons. We demonstrated that the rules it generated could be used to solve Raven's Advanced Progressive Matrices with similar performance levels to those observed in adult human subjects. By analyzing error patterns, we demonstrated that not only does the model achieve similar performance, when it fails in its reasoning it does so in a similar fashion to human subjects. We showed that the rules generalize to novel matrices, indicating that the model has correctly extracted general principles describing the matrix. And finally, we demonstrated that the same rule generation mechanisms can be applied across tasks.

The key cognitive feature of this model is its ability to dynamically generate rules describing patterns in its inputs, in a manner compatible with human performance and without relying on prior information. Previous models of Raven's problem solving have had difficulty accounting for the inductive generation of rules, thus the development of such a model is an important addition to our theoretical understanding of the RPM. In addition, this dynamic rule generation is a fundamental ability in not only the RPM but in a broad range of cognitive tasks—everything from visual pattern recognition, to syntactic processing, to empirical induction. The specific implementation of the system we have developed is designed for the structure of RPM problems, but the rule generation computations do not depend on that structure. All of the processing is based on the representations of Vector Symbolic Architectures, and implemented using the general mechanisms of spiking neurons. In the case of the RPM these VSA representations were used to describe the cells of the matrix, but they could also be used to describe parts of a sentence or lists of factual observations. Regardless of the underlying information described by the representations, the same processes can be used to extract rules describing the patterns in that information. Thus the dynamic rule generation system used in this model can be employed in other contexts and used to help understand a broad range of cognitive tasks.

It is important to note that this model complements rather than supersedes previous work; other models have strengths in other areas, and benefit from this available explanation of rule generation just as this model benefits from their work in domains such as visual processing. The work of Lovett et al. (2010) on automated relationship extraction and correspondence detection, and that of Kunda et al. (2013) on non-propositional representations, both tackle important aspects of RPM performance that are not captured by this model. We may look forward in the future to systems that combine the strengths of these different approaches into a unified model of these complex problem solving abilities.

One possible concern is that by building different rule generating components into the system (described in the Model architecture section), the model is subject to the same concern expressed in regard to previous models—that too much information is being built in by the modelers. In other words, could an analogy not be drawn between the rule generation components of our system and, for example, the rule productions of the Carpenter et al. (1990) model? The distinction between the two lies in the difference between induction and recognition. In our model, each rule component represents a general process that can generate any rule (of its respective type) that can be represented in VSAs. The components do not compare the inputs to known patterns, they generate each pattern from scratch based on the data they are given. For example, the sequence component takes a set of vectors as input, and generates a rule built only from the content of those vectors—it does not have access to any external information about what kinds of differences between cells correspond to what kinds of sequences. In addition, our rules are not restricted to the problem type of the RPM (geometric shapes in a 3×3grid); any information that can be represented in VSAs could be analyzed by these rule-finding systems (see Experiment 5 and Eliasmith et al. (2012) for demonstrations of the same mechanisms being applied to solve several different tasks).

Another key feature of this model is its biological basis. A computer program that can generate RPM rules computationally is interesting from an AI perspective, and can provide insight into high-level aspects of human performance, but it is limited in its ability to help us understand the inner workings of human cognition. That is why we use the Neural Engineering Framework to implement the model in biologically detailed networks of spiking neurons. This step serves two main purposes. First, it grounds the model in empirical data. By implementing the computations of the model in neural behavior, we ensure that those computations actually could be carried out by the hardware of the brain. The second advantage of the neural implementation is the ability to compare the results of the model to a broader range of human data. For example, in Experiments 6, 8, and 9 we were able to directly compare the neuron loss in human aging with neuron loss in the model, and in Experiment 7 we were able to combine those low-level neural manipulations with manipulations suggested by higher level fMRI data. Explorations such as this are only possible if a biologically detailed implementation is a fundamental feature of the model.

That being said, there are some important limitations to this model. This is a model of a specific aspect of cognition, dynamic rule generation, which is just one of the abilities needed to solve a Raven's matrix. Other abilities are equally important and challenging problems, but are not part of this model (e.g., visual processing) or are not modeled in the same neural detail as rule generation (e.g., the control system). This means that we must be careful when interpreting the model's results. The complete brain in a human solving a Raven's matrix is a far more complex system, which has far more dynamic and flexible performance, than this model. As our understanding of these other systems (and available computing power) increases, we will be able to build increasingly complex models that better capture the complete range of abilities in a given task. However, because this model is implemented in the general language of neural behavior, its features can be integrated into a more complete model without changing this model's fundamental implementation (as demonstrated in a simplified form by Eliasmith et al., 2012). Thus while a more complete model would better capture the full spectrum of human performance, we suspect that it would still include the features central to this model.

With respect to the neural basis of intelligence, we have used the specific domain of cognitive decline to demonstrate the potential insights that may be gained from a cognitively complex, biologically detailed model. Complex cognition has allowed us to examine performance on a difficult, real-world reasoning task (Raven's Advanced Progressive Matrices), while biological detail has allowed us to recreate neurophysiological changes observed in aging (neuron loss and representational dedifferentiation). We have demonstrated that these manipulations result in similar cognitive changes as those observed in aging human subjects, from raw performance to error patterns to individual differences. Note that we do not mean to suggest that these two factors account for all of the cognitive changes observed in aging. This model could be used to investigate many other aspects of aging, such as dendritic shrinkage (by eliminating incoming projections to the model neurons), loss of connectivity (by cutting connections between different neural populations), reduced neurotransmitter efficiency (by decreasing connection weights or introducing probabilistic firing), or pathologies such as Alzheimer's (containing a combination of factors). We chose the factors presented in this work because they are well-suited to demonstrating the value of models combining complex cognition with biological detail, through the model's capacity to empirically examine the links between neural processes and cognition.

In sum, the present model suggests novel, neurally plausible mechanisms for central aspects of complex cognition. We have demonstrated that these mechanisms are consistent with human cognitive performance, and how they can be manipulated to explore the functional impacts of neurophysiological changes. In general, models of this type are well suited to unifying our neural and cognitive characterizations of complex human behavior. While the present model is only an initial step in that direction, the general methods employed here should allow us to continue to extend such models to capture an increasingly broad range of relevant neural and cognitive phenomena.

Acknowledgments

This work was supported by the Natural Sciences and Engineering Research Council of Canada, CFI/OIT, Canada Research Chairs, and the Ontario Ministry of Training, Colleges, and Universities.

Appendix A. Neural implementations

A.1. The Neural Engineering Framework

VSAs allow us to represent and manipulate the information in a Raven's matrix at an abstract mathematical level, but we also want to implement those representations and operations at the neural level. The NEF provides a general framework for translating high level variables and operations into the activity of and connections between large networks of biologically plausible spiking neurons (Eliasmith &

Table A1
Main parameters of the model with default values.

Parameter	Value	Description
Vector dimension	30	Dimension of VSA vectors
Neurons per dimension	25	Number of neurons per VSA dimension
Vocabulary seed	–	Random seed for generating vocabulary
Correctness threshold (sequence/set/operation)	0.7/0.9/0.8	Minimum score indicating a correct rule has been found
Step size	0.2	Length of time (in seconds) to present each input
Vector similarity	1.0	Maximum similarity allowed in vocabulary
Similarity threshold (same/distinct)	1.0/0.9	Threshold to detect same/different features
Answer threshold	0.0	Minimum difference required to differentiate matrix answers
τ_{RC}	0.02	Leak rate of LIF neuron
τ_{ref}	0.002	Refractory period of LIF neuron
Max firing rate	200–500	Maximum firing rate of neurons
τ_{PSC}	0.007	Decay of post-synaptic current
Learning rate	0.4	Scale on input to averaging integrator
Forget rate	0.2	Rate at which information is lost in averaging integrator

Anderson, 2003). In our model we use this framework to represent the VSA vectors and carry out the VSA operations (binding and superposition) on those vectors.

The NEF represents information in a distributed manner, using the combined information from a population of neurons to represent a value (in this case, the VSA vectors). There are two important components to representation. The first is encoding a value into spikes—transforming the information contained in the input into the activity of the neurons (a to b in Fig. A1). The second component is decoding spikes into a value (c to d), so that it is possible for the modeler to interpret what information is being represented by the population.

Encoding a vector $x(t)$ (all computations occur over time, t) into the spike train of neuron a_i is accomplished through a neuron model

$$a_i(x(t)) = G_i\left[\alpha_i e_i x(t) + J_i^{bias}\right] \tag{4}$$

which describes the activity of neuron a_i as a function of its input current. G_i is a function representing the nonlinear neuron characteristics. It takes a current as input (the value within the brackets), and uses a model of neuron behavior to output spikes. In our model we use leaky integrate-and-fire neurons, which strike a balance between computational simplicity and realistic neural dynamics. However, this formulation allows any neuron model to be substituted for G_i, so the neuron dynamics can be made as simple or detailed as desired without changing the overall model. The variables α_i, J_i^{bias}, and e_i are the parameters of neuron α_i. The parameters α_i and J_i^{bias} do not directly play a role in the encoding of information, but rather are used to provide variability in the firing characteristics of neurons. This allows the modeler to capture the heterogeneity observed in biological neurons. The parameter e_i represents the neuron's preferred stimulus—which inputs will make it fire more strongly. This is an important factor in the neuron's firing, as it differentiates what properties of the input a neuron will respond to. Specifically, the dot product between e_i and the input (i.e., their similarity) drives a particular cell. In summary, the activity of neuron a_i is a result of its unique response (determined by its preferred stimulus, e_i) to the input $x(t)$, passed through a nonlinear neuron model in order to generate

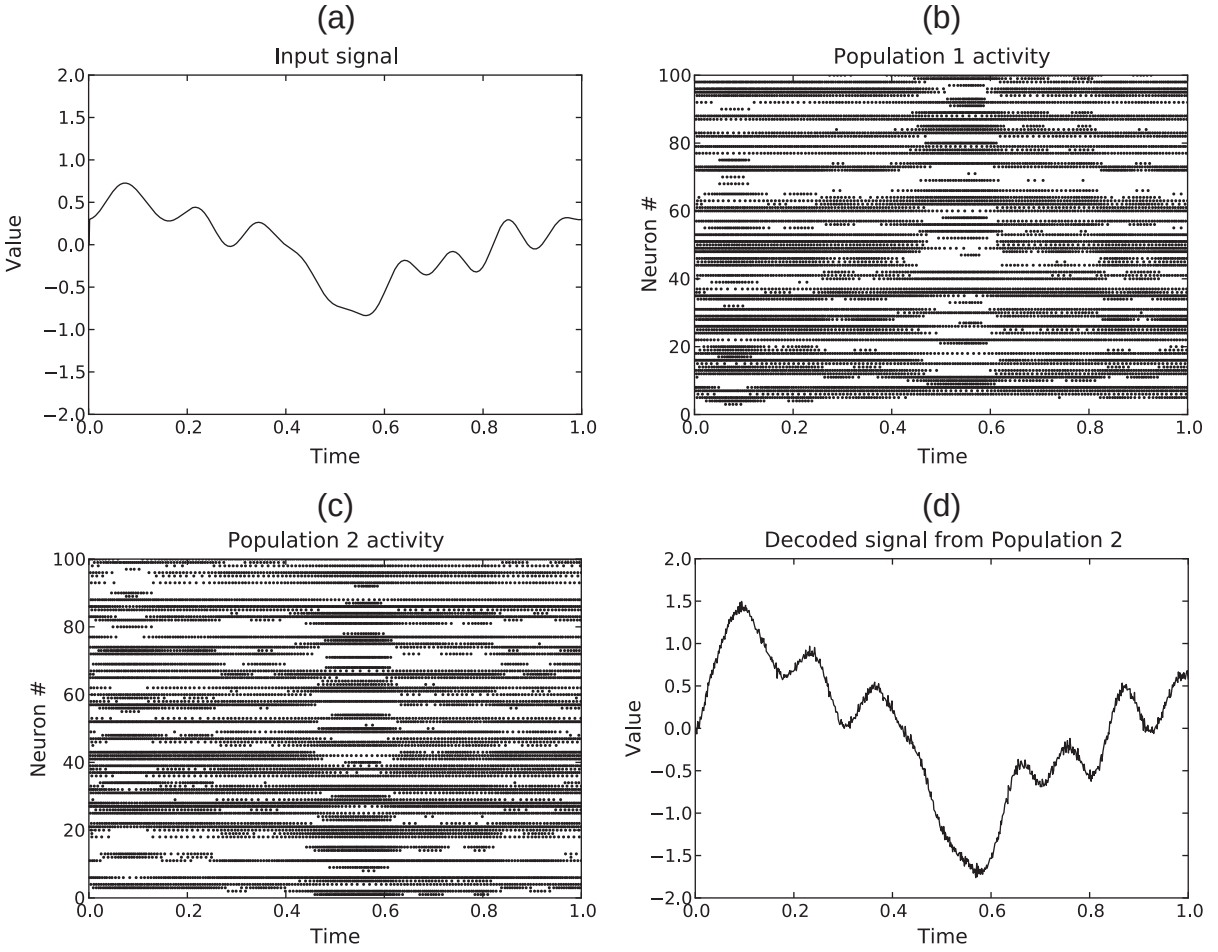

Fig. A1. Recordings from a simple network in which an input signal (a) is fed into a population of simulated spiking neurons (b). That population is then connected to a second population (c), with the connection weights calculated to double the represented value. Finally, the spiking activity of the second population is decoded back into a value (d). Note: b and c are spike rasters; each row corresponds to one neuron, and each dot indicates that the neuron fired a spike at that time.

spikes. This characterizes the encoding of a signal (e.g., a VSA vector) into neural spikes.

The second, equally important, part of neural representation is the reverse process: decoding the activity of a population into a value, $\hat{x}(t)$. This is accomplished through the formula

$$\hat{x}(t) = \sum_i h(t) * a_i(x(t))d_i \tag{5}$$

where $*$ denotes standard (not circular) convolution. Essentially this is modeling the unweighted current that would be induced in the post-synaptic cell by the spikes coming out of a_i (we will discuss how the synaptic weights are determined shortly). The $a_i(x(t))$ are the spikes generated in Eq. (4). The function $h(t)$ is a model of the post-synaptic current generated by each spike; convolving that with $a_i(x(t))$ gives the total current generated by the spikes from a_i. The d_i parameters are the optimal linear decoders, which are calculated analytically so as to provide the best linear representation of the original input $x(t)$. The equation for determining the decoders is as follows:

$$d = \Gamma^{-1}\Upsilon$$

where

$$\Gamma_{ij} = \int a_i(x)a_j(x)dx$$
$$\Upsilon_j = \int a_j(x)f(x)\ dx. \tag{6}$$

The variable $a_i(x)$ denotes the average firing rate of neuron a_i given input x. Note that in this case the temporal aspect, t, has been removed. This is because only the average firing rate of the neuron for a given input is used, rather than the specific timing of the spikes. However, because the two measures (firing rate and spike timing) arise from the same neuron model, the decoders calculated based on rates can be applied in the temporal case. The function $f(x)$ is the operation to be performed on the represented value; if the goal is to decode the value represented by the population without modification, as is usually the case, then $f(x) = x$. For the mathematical derivation showing the optimality of the resulting decoders, see Eliasmith and Anderson (2003).

Roughly speaking, the decoders can be thought of as creating a mapping from the output current onto the input value that created that output. Since the input was transformed into current using the nonlinear neuron model, it is impossible to recreate the input perfectly using linear decoders. Thus the resulting value is an approximation, $\hat{x}(t)$, of the original input, $x(t)$. That is why we use large populations of neurons, with heterogeneous properties—so that the combined activity of all the neurons in the population "balances out" the inaccuracy in any one neuron. Given enough neurons it is possible to get a very accurate recreation of the original input; for a detailed analysis of the link between population size, neuron properties, and representational accuracy, see Eliasmith and Anderson (2003).

In addition to representing VSA vectors, we also need to carry out transformations—the VSA operations—on those vectors. We will examine several different types of transformations, and then demonstrate how they can be combined to support all the calculations required to implement VSAs.

The first transformation is a scale on the input (e.g., calculating $y = 2x$). Assuming that there are two populations a and b which will represent the x and y values, respectively, then this amounts to a question of how to set the weights on the synaptic connections between a and b. Referring back to Fig. A1, the connection weights between the two populations—1(b) to 1(c)—is what needs to be determined. The goal is to set the weights in such a way that when the neurons in population a are firing to represent x, this will cause the neurons in population b to fire in such a way that they represent $2x$. Recall that the spiking of neuron b_j is a result of its nonlinear response (G_j) to the input current. However, the input to population b is no longer a direct value, but is instead the output from population a. The output of population a is given by Eq. (5), so to calculate the firing of population b, Eq. (5) can be substituted in for $x(t)$ in Eq. (4):

$$\begin{aligned} b_j(x(t)) &= G_j\left[\alpha_j e_j\left(\sum_i h(t) * a_i(x(t))d_i\right) + J_j^{bias}\right] \\ &= G_j\left[\sum_i h(t) * a_i(x(t))\alpha_j e_j d_i + J_j^{bias}\right] \\ &= G_j\left[\sum_i h(t) * a_i(x(t))\omega_{ji} + J_j^{bias}\right]. \end{aligned} \tag{7}$$

In other words, the input current of neuron b_j is equal to the output current of all the a neurons connected to b_j, multiplied by the connection weights. In most neural simulations the connection weights, ω, need to be learned. The NEF formulation allows the connection weights to be analytically determined: $\omega_{ji} = \alpha_j e_j d_i$.[10] Referring back to the descriptions of the variables in Eqs. (4) and (5), what this means is that the connection weight between neurons a_i and b_j is equal to the preferred stimulus of b_j multiplied by the decoders for a_i (all scaled by the gain, α_j, on b_j). So far the value has not actually been transformed, the output of a is simply being passed directly to b. In order to calculate the transformation $y = 2x$, the connection weights are multiplied by 2: $\omega_{ji} = 2\alpha_j e_j d_i$. More generally, if we want to multiply the vector x by the matrix C, we can calculate the weights as $\omega_{ji} = \alpha_j e_j C d_i$.

The second transformation is combining two values ($z = x + y$). This is almost identical to Eq. (7), except that now the input current is coming from two populations instead of one:

$$c_k(x(t) + y(t)) = G_k\left[\sum_i h(t) * a_i(x(t))\omega_{ki} + \sum_j h(t) * b_j(y(t))\omega_{kj} + J_k^{bias}\right] \tag{8}$$

where $\omega_{ki} = \alpha_k e_k d_i$ and $\omega_{kj} = \alpha_k e_k d_j$. A second, analogous term has been added to incorporate the second input population. This can be continued in the same way to combine any number of inputs.

The final step is to combine the previous two elements to compute arbitrary transformations of the form $z = C_1x + C_2y$. This is simply Eq. (8), except $\omega_{ki} = \alpha_k e_k C_1 d_i$ and $\omega_{kj} = \alpha_k e_k C_2 d_j$. Thus with these techniques any linear transformation can be computed using simulated spiking neurons.

[10] This does not mean that the connection weights could not be learned instead; calculating them analytically simply allows us to skip the lengthy training process. For a demonstration of using a realistic spiking neural learning rule to achieve the same result, see Bekolay & Eliasmith (2011).

Notice that the linear transformation discussion has been solely about encoding from value to spikes, there has been no discussion of decoding from spikes to values. This is because the decoding is unaffected by these linear transformations. Recall that the optimal linear decoders map from output current to the input value that caused that current. These manipulations are only changing the input—the mapping remains the same. However, it is also possible to compute different mappings, by setting $f(x)$ to different functions in Eq. (6). This makes it possible to approximate nonlinear functions such as multiplication. Multiplication is particularly relevant for VSAs, because circular convolution is based on the multiplication of vector elements (see Eq. (1)).

There are a number of methods to perform multiplication ($z = xy$) using spiking neurons (Eliasmith & Anderson, 2003; Polsky, Mel, & Schiller, 2004). Here we employ a technique that uses only linear dendrites, as this is the most conservative assumption regarding dendritic computation. With this approach, multiplication involves a two stage process, using an intermediate population M. The M populations takes x and y as input and combines them into a two dimensional value $m = [x\ y]$. This is a linear transformation, and so is accomplished as described above. The decoders of M, instead of mapping onto the same two dimensional space as the input, map onto a one dimensional space by setting $f(m) = m_1 \times m_2$ in Eq. (6). Whereas before the decoders were used to directly translate output current into the value that caused that current, now they are adding a transformation to that translation. The decoders are still only linear weights and so only approximate a nonlinear transformation such as multiplication, but they can do so sufficiently well for our purposes (the accuracy of the approximation is proportionate to the number of cells, so the multiplication could be made arbitrarily accurate given an increasing number of cells). Together with linear transformations, these nonlinear approximations are sufficient to carry out all of the VSA operations.

A.2. Sequence component

There are two primary computations in the neural implementation of this component (see Fig. 6): calculating the transformation between two cells (i.e., calculating $A' \circledast B$), and averaging across those transformations. The key aspect of the first computation is performing circular convolution, as calculating the pseudoinverse (A') is a linear transformation (see Eq. (2)). Circular convolution (Eq. (1)) is a complex operation with a large number of nonlinear operations (multiplication), but it can be made much simpler by noting that it is equivalent to an element-wise product in the frequency domain (Plate, 2003). Thus we can compute the circular convolution of two vectors by transforming those vectors into the frequency domain using a Discrete Fourier Transform (DFT; a linear operation, and so performed in neurons via Eq. (7)), calculating the element-wise product (using the multiplication method described above), and then transforming the result back using the inverse DFT. This dramatically reduces the number of multiplications that need to be performed, allowing for a sufficiently accurate implementation of circular convolution in spiking neurons.

The next challenge is averaging the individual transformations. The standard way of calculating an average is to sum all the items and divide by the number of items being averaged. However, this assumes that all those computations are occurring simultaneously and instantaneously. In a real brain this is not the case; computations are occurring over time, thus the system needs to remember the results of previous calculations and combine them with new results. Therefore we instead use a running average; as each new transformation is calculated for a given pair of cells, it is used to update a stored value so that over time the stored value approaches the average of its inputs. This is expressed by the formula

$$T_{i+1} = fT_i + l(A_i' \circledast B_i) \tag{9}$$

where T_i is the average of the first i transformations, f is the rate at which previous information is forgotten, and l is the emphasis placed on new information. If we set $f = 1 - l$ and $x = 1/(i + 1)$ then this formula calculates an exact average.

We implement this formula in neurons by using a modified integrator. An integrator is a population of neurons with recurrent connections to itself that when given no input will maintain its current value over time, acting like working memory (Eliasmith, 2005). In this case we use the integrator to represent the T_i values—the running average of the individual transformations calculated by the circular convolution network. A basic integrator simply sums its inputs rather than averaging them, so we modify the integrator by adding a scale to the input (l) and causing it to constantly forget past information (f). Fig. A2 shows how Eq. (9) maps onto the components of the integrator. As mentioned, calculating an exact average requires setting $l = 1 / (i + 1)$ —in other words, changing the values of l and f as each new item is presented. In the integrator, this would require very fast and accurate changes on many neuronal connections. This is not biologically plausible, so instead we pick fixed values for l and f that approximate a true average. This results in a less accurate average, but it is a more plausible mechanism and, because the values being averaged have significant noise anyway, is not a critical impediment to performance.

Once all the individual transformations have been calculated and input to the integrator, the value stored will be the average rule for the whole matrix. This value is then

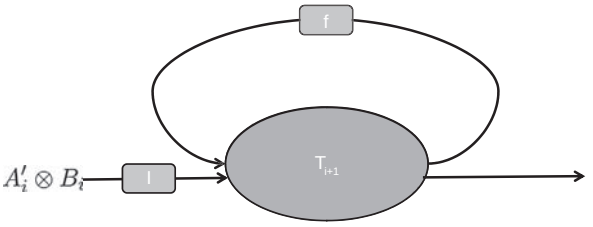

Fig. A2. Schematic diagram of the neural population implementing averaging (Eq. (9)). The population receives two inputs: the current transformation ($A_i \circledast B_i$) and the overall transformation from the previous timestep (T_i, delivered through the recurrent connection). These inputs are weighted by the scaling factors l and f, respectively. The population sums these two inputs, calculating T_{i+1}, the new overall transformation.

convolved with the vector for the second last cell ($cell_{3,2}$). This is done using an identical circular convolution network as was used to calculate the individual transformations, but without the pseudoinverse. The resulting value is the component's hypothesis as to the contents of the blank cell.

A.3. Set component

The set component is similar to the sequence component in that in both cases the system needs to calculate several particular pieces of information and then average across them (see Fig. 7). In the sequence case the particular information was the transformation between two cells, in this case it is the superposition of the three cells in a row.

Recall that in the Holographic Reduced Representation (Plate, 2003) implementation of VSAs, superposition is simply vector addition. We implement this at the neural level by having a population storing each of the values we want to sum (the vector representation of each of the three cells in the row). We then combine the output of these three populations into a single population using Eq. (8), which will then be representing the sum of its inputs—the set representation for those three items. The averaging is then performed in exactly the same way as it was in the sequence component, using a working memory-like population. The result is that the population will be storing an average of the two row sets, which is the general set rule for the matrix.

To select an answer we use two more populations storing the vectors for the two known cells in the third row, combined with the integrator population storing the overall rule. We then connect those three outputs to a final population in the same way as we did for the initial summation, except that we scale the output of the first two populations by -1 (Eq. (7)). The result is that the final population will receive as input the overall matrix set minus the two known cells in the third row, and thus will be storing the remaining items in the set—the contents of the blank cell.

A.4. Operation component

The operation component consists of two parallel systems: one to work on the set of common features, and one to work on the set of distinct features (see Fig. 9). The two systems are nearly identical, so we will describe only one of them and point out any differences.

The neural calculations begin with the formation of the combined vector representing either the summation or subtraction of the first two cells (to calculate the common or distinct features, respectively). This is a matter of combining the input from two populations as in Eq. (8); to perform the subtraction, one of the populations' output is scaled by -1.

After forming the combined vector, the next step is to perform the similarity calculations—determining the inner product of the combined vector and each of the vectors in the vocabulary. This is done in our model by creating a neural population to represent each vector in the vocabulary. Each population then calculates the inner product (a linear transformation) between the vector it represents and its input, the combined vector.[11]

The key aspect of determining the set of combined/distinct features is the thresholding. This acts as a gate on the output of the vocabulary vector populations, so that any population whose calculated inner product does not surpass the threshold has an output of zero. This is a nonlinear operation, and so is performed by modifying $f(x)$ in the decoder calculations (Eq. (6)). We set $f(x)$ to be the step function[12]

$$f(x) = \begin{cases} x & : x \geq threshold \\ 0 & : x < threshold \end{cases}. \tag{10}$$

The result is that populations will output zero when the similarity is beneath the threshold, and the vector they represent when the similarity exceeds the threshold. The outputs of all the vocabulary populations are then combined into a single population. The populations with zero output effectively contribute nothing, so the result is that the output population represents the sum of all the vectors whose similarity exceeded the threshold. In other words, the population represents the set of features that have been determined to be common (or distinct).

Having calculated the set of common or distinct features between the first two cells of the row, the next step is to determine the similarity between the common (or distinct) features and the third cell, in order to generate the rule. Recall that similarity is defined as the inner product of the two vectors, which we calculate by taking the element-wise product (using the multiplication method described in the discussion of the NEF) and then summing across the dimensions of the result (a linear operation). The rules are then averaged across the rows using the same integrator averaging method described in Fig. A2 to create a general rule for the whole matrix.

The final step is to generate a hypothesis for the blank cell. To begin, the set of common/distinct features for the first two cells of the third row are calculated using the system described above. The output of these systems (vectors representing the set of common and distinct features) is then weighted by the general rule for the matrix (another element-wise product). The outputs from the populations representing the two weighted sets are then combined into a final population, which represents the system's prediction for the blank cell.

A.5. Parameter settings

Typical parameter settings for the model are listed in Table A1. In the description of the experiments, these are the values used unless explicitly mentioned otherwise. In experiments involving a population of models, the standard population size is 30 model instances. The vocabulary words and neuron tunings (the e_i in Eq. (4)) are randomly generated. They are d dimensional vectors, where d is the

[11] This has been shown to scale to adult-sized vocabularies by Stewart et al. (2011).

[12] In the system to detect distinct features the similarity can be negative due to the subtraction, and so the function is changed to $f(x) = \begin{cases} |x| & : |x| \geq threshold \\ 0 & : |x| < threshold \end{cases}$.

vector dimension specified in Table A1. The vector elements are chosen from $\mathcal{N}(0, 1)$, then the vectors are normalized to be unit length.

Appendix B. Representing Raven's matrices using VSAs

As mentioned in the discussion of VSAs, we use an attribute-value system to encode Raven's matrices. Each cell of the matrix is encoded as a separate vector, so there are eight vectors describing the matrix (we do not bother encoding the blank cell). The answers are encoded in exactly the same way as the cells of the matrix, for a total of 16 vectors to describe the information in each matrix.

The vocabulary is fixed and common for the entire RPM test, we do not generate new vocabularies tailored to each matrix (as is done in Carpenter et al., 1990). The complete list of attributes and values used to describe the matrices is given in Table B1.[13] This vocabulary allows us to describe all of the problems in the RPM. However, the model does not make any assumptions about how the matrices are described; any alternate vocabulary that captures equivalent information could be substituted into the system without requiring modification to the model itself.

The descriptiveness of the vocabulary is limited by the dimension of the vectors. Recall from Fig. 2 that the more words in the vocabulary, the higher the dimension of the vectors required to represent those words. In our model we are restricted by computational limits to 30 dimensional vectors, and a correspondingly simple vocabulary. Thus for certain obscure values (such as strange non-regular shapes) we might not have a corresponding word in the vocabulary. In those cases we substitute in a value that is in the vocabulary, leaving the structure of the problem unchanged. Human subjects will be able to develop a much richer and more nuanced description of the matrices, as they could employ up to an estimated 500 dimensions (Eliasmith, 2013). However, as mentioned, the model is not dependent on this vocabulary; as increasing computational power allows us to build more detailed vocabularies, those vocabularies can be incorporated into the model without changing the model itself.

In most cases the vocabulary vectors are randomly generated, as in the standard procedure for VSAs. However, for certain values a more complex generation method is required. These are values that are related to each other in some way, such as numbers or locations. The concepts of *north* and *north-west* are related to each other innately by the idea that *north-west* is to the *west* of *north*. Similarly, *two* is related to *one* by the idea that *two* is one more than *one*. This is in contrast to an attribute such as shape, where values like *square*, *cross*, and *dot* have no particular relationship to one another. When the values are related we want to capture those relationships in our VSA vocabulary, and so the vectors cannot be random. Instead, we generate a base vector (e.g., a vector *one*) and an "increment" vector for that attribute. Then the concept for *two*, instead of being randomly generated, is created by convolving *one* with the increment vector. Convolving *one* with the increment vector twice will give *three*, and so on. Evidence for the existence of these kinds of representations in human subjects is explored in detail in Carey (2009); in particular, she examines how the understanding of the "increment" relationship linking the natural numbers develops in young children (for earlier work see Gelman & Gallistel, 1978).

Table B1
Vocabulary used to describe the Raven's matrices.

Attribute	Example values
Shape	Circle, square, diamond, …
Number	One, two, three …
Linestroke	Normal, dashed, bold …
Angle	0°, 45°, 90°, …
Length	Short, medium, long
Width	Short, medium, long
Location	NW, N, NE, … (quadrant of the cell)
Shading	None, solid, left-hatch, …
Linetype	Straight, curved, wavy, …
Radialpos	Inner, middle, outer
Portion	Left-half, right-half, bottom-half, top-half
Skew	Left, none, right
Misc	Present, absent

To describe a cell of the matrix we create sets of these attribute-value pairs through the VSA superposition operation. In general, we only include attributes that are relevant to the problem; attributes that are constant across all rows and columns are left out of the description. This is consistent with the Carpenter et al.'s (1990) approach, upon which ours is loosely based. However, if an attribute is particularly important we will include it even if it is not relevant. For example, to describe $cell_{1,1}$ of the matrix shown in Fig. 1 we would create the vector *shape* ⊛ *arrow* + *number* ⊛ *one* + *angle* 90°. *Shape* ⊛ *arrow* is constant throughout the matrix so is not necessary to solve the problem, but we include it anyway as otherwise we are left with a strange description of the number and angle of an unknown object. However, we would not include an attribute such as line-stroke, as the boldness of the lines is not important in this problem. This step is not critical to the model's successful performance; if non-relevant attributes were left in, they would simply be described via a constant sequence or set rule. However, this is not an accurate reflection of how human subjects solve these problems; Carpenter et al. (1990) describe how subjects instinctively perform a similar filtering process, ignoring features that are constant throughout the matrix. Thus in order to have the model's rule generation conform as closely as possible to human rule generation, we also leave out these irrelevant attributes.

One added complication occurs when there is more than one feature present in the cell. For example, naively we might describe $cell_{1,1}$ of Fig. 3 by *shape* ⊛ *circle* + *shading* ⊛ *solid* + *number* ⊛ *one* + *shape* ⊛ *arrow* + *shading* ⊛ *none* + *angle* ⊛ 90°. However, the superposition operation does not preserve any order information, so it is unclear in that description which of the attributes belongs to which feature. The vector could equally well be describing a shaded arrow and unshaded circle. To resolve this ambiguity we add a final component into the vector which acts as a "tag" for each feature (Plate, 2003). The tag is equal to the convolution of all the attribute-value pairs for that feature. For example, the tag for the shaded circle would

[13] The "misc" attribute is used to describe attributes that are only present in single matrices and take on a binary value of either being present or not. For example, in one matrix shapes can either be folded in half or not. Rather than creating a separate attribute for "foldedness", we would capture this under the "miscellaneous" attribute.

be $A = shape \circledast circle \circledast number \circledast one \circledast shading \circledast solid$, and a similar B tag for the arrow. The complete description of the cell is then $shape \circledast circle + shading \circledast solid + number \circledast one + shape \circledast arrow + shading \circledast none + angle \circledast 90° + A + B$. This description preserves all the information about the items in the cell, but is no longer ambiguous. It will not be the same as the vector for a shaded arrow and unshaded circle, because the tag for unshaded circle ($shape \circledast circle \circledast number \circledast one \circledast shading \circledast none$) is not present in the description of the cell.

Appendix C. Working memory

The Raven's Progressive Matrices test has a long association with research into working memory. In general, fluid intelligence, the ability most often associated with the RPM, has been shown to be strongly correlated with many working memory tasks (Conway et al., 2003), even leading some to argue that the two are almost synonymous (Kyllonen & Christal, 1990). More specifically, one of the main contributions of the empirical and modeling work of Carpenter et al. (1990) was investigating the relationship between working memory and RPM performance. Thus, while working memory is not the focus of this paper, we provide a brief discussion here in order to illustrate how this model might be used to connect to that body of work. In particular, Carpenter et al. (1990) suggested that differences in working memory explain significant portions of individual differences in RPM performance. This is the question we will address here: can the performance of the model on the RPM be explained by its underlying working memory ability?

To test this we began by setting up a population of 300 models as in Experiment 9. We then ran the models through the Raven's test as normal to get a measure of their individual ability. Next, in order to get a measure of each individual's working memory ability, we ran them through the RPM again but introduced a delay between when the model generates a rule and when it picks an answer of 1–10 s. This forces the model to maintain the rule it has generated in the averaging integrator populations (see the model descriptions). This is the population that stores the individual rules that are calculated and updates the stored information as new data is encountered, a classic example of the type of processing attributed to working memory. In addition, recurrent neural attractors such as this are frequently used as models of working memory in computational neuroscience (Brody, Romo, & Kepecs, 2003; Laing & Chow, 2001). Thus it is reasonable to describe the performance of this component as a measure of working memory ability.

In order to assess the performance of the working memory component we calculated the average similarity between the rule vectors stored in this component at the beginning and end of the delay periods, across all 36 items of the RPM. Successful performance involves minimizing any drift in the stored value over the delay period, so higher working memory ability corresponds to greater similarity between the before and after rules.

Next, we looked at the correlation between each model's working memory score and overall RPM score. The result showed no correlation between the two measures ($r = 0.008$ for a 1 second delay, $r = -0.068$ for a 10 second delay). In other words, it appears that the overall performance of the models cannot be explained by the performance of the working memory component. This seems surprising given the known link between working memory and the RPM, but in fact it is consistent with current data if we look in more detail at what aspect of working memory is being assessed.

What we are measuring is the very basic ability of the model to store a rule, with no modification, over the delay period. This capacity for storage is often separated in the working memory literature from the more complex ability to manipulate and manage the contents of that storage (Conway, Kane, & Engle, 2003; Kane & Engle, 2002). Work with the RPM has revealed that it is the former, not the latter, that best explains the observed correlations between the RPM and working memory (Unsworth & Engle, 2005; Wiley, Jarosz, Cushen, & Colflesh, 2011). Thus researchers have argued that it is variation in the ability to control working memory that contributes to individual differences on this task, not variations in storage capacity or quality. Our results are in line with this perspective, demonstrating that although the ability to store information is important to task performance, variations in this ability do not explain individual differences.[14]

This leaves open the question of whether or not the model would demonstrate correlations between performance and more complex aspects of working memory. In addition, other work on the RPM has suggested that the observed working memory correlations may in fact be a result of a complex cascade involving more fundamental components such as processing speed or learning ability (Tamez, Myerson, & Hale, 2008, 2012). Future work in this regard might involve manipulations to the upstream rule generation mechanisms, or to the controller that coordinates the three neural components. As mentioned at the outset, we do not seek to address the complex field of working memory in this paper. We intend this discussion only as a brief demonstration that this model can be used to connect to that body of work on the RPM.

References

Ashby, F. G., Ennis, J. M., & Spiering, B. J. (2007). A neurobiological theory of automaticity in perceptual categorization. *Psychological Review*, *114*, 632–656.

Babcock, R. (2002). Analysis of age differences in types of errors on the Raven's advanced progressive matrices. *Intelligence*, *30*, 485–503.

Bekolay, T., & Eliasmith, C. (2011). A general error-modulated STDP learning rule applied to reinforcement learning in the basal ganglia. In A. Churchland, & B. Mel (Eds.), *Proceedings of the 8th computational and systems neuroscience meeting, COSYNE 2011, Salt Lake City*.

Bolin, B. J. (1955). A comparison of Raven's progressive matrices (1938) with the ACE psychological examination and the Otis Gamma mental ability test. *Journal of Consulting Psychology*, *19*, 400.

Bor, D., Duncan, J., Wiseman, R. J., & Owen, A. M. (2003). Encoding strategies dissociate prefrontal activity from working memory demand. *Neuron*, *37*, 361–367.

Bors, D., & Stokes, T. (1998). Raven's advanced progressive matrices: norms for first-year university students and the development of a short form. *Educational and Psychological Measurement*, *58*, 382–398.

Bourjaily, M., & Miller, P. (2011). Excitatory, inhibitory, and structural plasticity produce correlated connectivity in random networks trained

[14] The Carpenter et al. (1990) model is somewhat ambivalent in this regard, as they found that both working memory control and working memory capacity were important to their model's performance.

to solve paired-stimulus tasks. *Frontiers in Computational Neuroscience*, *5*, 37.
Brody, C. D., Romo, R., & Kepecs, A. (2003). Basic mechanisms for graded persistent activity: discrete attractors, continuous attractors, and dynamic representations. *Current Opinion in Neurobiology*, *13*, 204–211.
Carey, S. (2009). Where our number concepts come from. *Journal of Philosophy*, *106*, 220–254.
Carpenter, P., Just, M., & Shell, P. (1990). What one intelligence test measures: A theoretical account of the processing in the Raven progressive matrices test. *Psychological Review*, *97*, 404–431.
Christoff, K., Prabhakaran, V., Dorfman, J., Zhao, Z., Kroger, J. K., Holyoak, K. J., et al. (2001). Rostrolateral prefrontal cortex involvement in relational integration during reasoning. *NeuroImage*, *14*, 1136–1149.
Conway, A. R. A., Kane, M. J., & Engle, R. W. (2003). Working memory capacity and its relation to general intelligence. *Trends in Cognitive Sciences*, *7*, 547–552.
Deary, I. J., Corley, J., Gow, A. J., Harris, S. E., Houlihan, L. M., Marioni, R. E., et al. (2009). Age-associated cognitive decline. *British Medical Bulletin*, *92*, 135–152.
Deary, I. J., Whalley, L., Lemmon, H., Crawford, J., & Starr, J. M. (2000). The stability of individual differences in mental ability from childhood to old age: Follow-up of the 1932 Scottish Mental Survey. *Intelligence*, *28*, 49–55.
Der, G., Allerhand, M., Starr, J. M., Hofer, S. M., & Deary, I. J. (2010). Age-related changes in memory and fluid reasoning in a sample of healthy old people. *Aging, Neuropsychology, and Cognition*, *17*, 55–70.
Duncan, J., Burgess, P., & Emslie, H. (1995). Fluid intelligence after frontal lobe lesions. *Neuropsychologia*, *33*, 261–268.
Duncan, J., & Owen, A. M. (2000). Common regions of the human frontal lobe recruited by diverse cognitive demands. *Trends in Neurosciences*, *23*, 475–483.
Eliasmith, C. (2005). A unified approach to building and controlling spiking attractor networks. *Neural Computation*, *17*, 1276–1314.
Eliasmith, C. (2013). *How to build a brain: A neural architecture for biological cognition.* Oxford: Oxford University Press.
Eliasmith, C., & Anderson, C. (2003). *Neural engineering: Computation, representation, and dynamics in neurobiological systems.* Cambridge: MIT Press.
Eliasmith, C., Stewart, T. C., Choo, X., Bekolay, T., DeWolf, T., Tang, Y., et al. (2012). A large-scale model of the functioning brain. *Science*, *338*, 1202–1205.
Evans, T. (1968). A program for the solution of a class of geometric-analogy intelligence-test questions. In M. Minsky (Ed.), *Semantic information processing* (pp. 271–353). Cambridge: MIT Press.
Fjell, A. M., Walhovd, K. B., Reinvang, I., Lundervold, A., Salat, D., Quinn, B. T., et al. (2006). Selective increase of cortical thickness in high-performing elderly—structural indices of optimal cognitive aging. *NeuroImage*, *29*, 984–994.
Forbes, A. R. (1964). An item analysis of the advanced matrices. *British Journal of Educational Psychology*, *34*, 223–236.
Frank, M. J., & Badre, D. (2012). Mechanisms of hierarchical reinforcement learning in corticostriatal circuits 1: computational analysis. *Cerebral Cortex*, *22*, 509–526.
Gayler, R. W. (2003). Vector symbolic architectures answer Jackendoff's challenges for cognitive neuroscience. In P. Slezak (Ed.), *ICCS/ASCS international conference on cognitive science* (pp. 133–138). Sydney: University of New South Wales.
Gelman, R., & Gallistel, C. (1978). *The child's understanding of number.* Pennsylvania: Harvard University Press.
Gentner, D. (1983). Structure-mapping: A theoretical framework for analogy. *Cognitive Science*, *7*, 155–170.
Goh, J. O. (2011). Functional dedifferentiation and altered connectivity in older adults: Neural accounts of cognitive aging. *Aging and Disease*, *2*, 30–48.
Golde, M., von Cramon, D. Y., & Schubotz, R. I. (2010). Differential role of anterior prefrontal and premotor cortex in the processing of relational information. *NeuroImage*, *49*, 2890–2900.
Gray, J. R., Chabris, C. F., & Braver, T. S. (2003). Neural mechanisms of general fluid intelligence. *Nature Neuroscience*, *6*, 316–322.
Gurney, K., Prescott, T. J., & Redgrave, P. (2001). A computational model of action selection in the basal ganglia. *Biological Cybernetics*, *84*, 401–423.
Hazy, T. E., Frank, M. J., & O'Reilly, R. C. (2007). Towards an executive without a homunculus: Computational models of the prefrontal cortex/basal ganglia system. *Philosophical Transactions of the Royal Society of London. Series B, Biological Sciences*, *362*, 1601–1613.
Jung, R. E., & Haier, R. (2007). The Parieto-frontal integration theory (P-FIT) of intelligence: Converging neuroimaging evidence. *Behavioral and Brain Sciences*, *30*, 135–187.
Kaasinen, V., Vilkman, H., Hietala, J., Nå gren, K., Helenius, H., Olsson, H., et al. (2000). Age-related dopamine D2/D3 receptor loss in extrastriatal regions of the human brain. *Neurobiology of Aging*, *21*, 683–688.
Kane, M. J., & Engle, R. W. (2002). The role of prefrontal cortex in working-memory capacity, executive attention, and general fluid intelligence: An individual-differences perspective. *Psychonomic Bulletin and Review*, *9*, 637–671.
Kaufman, A., Reynolds, C., & McLean, J. (1989). Age and WAIS-R intelligence in a national sample of adults in the 20- to 74-year age range: A cross-sectional analysis with educational level controlled. *Intelligence*, *13*, 235–253.
Kirby, J. (1983). Effects of strategy training on progressive matrices performance. *Contemporary Educational Psychology*, *8*, 127–140.
Kroger, J. K., Sabb, F. W., Fales, C. L., Bookheimer, S. Y., Cohen, M. S., & Holyoak, K. J. (2002). Recruitment of anterior dorsolateral prefrontal cortex in human reasoning: a parametric study of relational complexity. *Cerebral Cortex*, *12*, 477–485.
Kunda, M., McGreggor, K., & Goel, A. K. (2013). A computational model for solving problems from the Raven's progressive matrices intelligence test using iconic visual representations. *Cognitive Systems Research*, *22–23*, 47–66.
Kunda, M., McGreggor, K., & Goel, A. K. (2012). Reasoning on the Raven's advanced progressive matrices test with iconic visual representations. In N. Miyake, D. Peebles, & R. P. Cooper (Eds.), *Proceedings of the 34th annual meeting of the Cognitive Science Society* (pp. 1828–1833). Austin: Cognitive Science Society.
Kyllonen, P. C., & Christal, R. E. (1990). Reasoning ability is (little more than) working-memory capacity?! *Intelligence*, *14*, 389–433.
Laing, C. R., & Chow, C. C. (2001). Stationary bumps in networks of spiking neurons. *Neural Computation*, *13*, 1473–1494.
Lee, K. H., Choi, Y. Y., Gray, J. R., Cho, S. H., Chae, J. H., Lee, S., et al. (2006). Neural correlates of superior intelligence: Stronger recruitment of posterior parietal cortex. *NeuroImage*, *29*, 578–586.
Lovett, A., & Forbus, K. (2012). Modeling multiple strategies for solving geometric analogy problems. *Proceedings of the 34th annual meeting of the Cognitive Science Society*.
Lovett, A., Forbus, K., & Usher, J. (2010). A structure-mapping model of Raven's progressive matrices. In S. Ohlsson, & R. Catrambone (Eds.), *Proceedings of the 32nd annual conference of the Cognitive Science Society* (pp. 2761–2766). Austin: Cognitive Science Society.
Madden, D. J., Bennett, I. J., & Song, A. W. (2009). Cerebral white matter integrity and cognitive aging: Contributions from diffusion tensor imaging. *Neuropsychology Review*, *19*, 415–435.
Marshalek, B., Lohman, D., & Snow, R. (1983). The complexity continuum in the radex and hierarchical models of intelligence. *Intelligence*, *7*, 107–127.
Melrose, R. J., Poulin, R. M., & Stern, C. E. (2007). An fMRI investigation of the role of the basal ganglia in reasoning. *Brain Research*, *1142*, 146–158.
Morse, C. K. (1993). Does variability increase with age? An archival study of cognitive measures. *Psychology and Aging*, *8*, 156.
Narr, K. L., Woods, R. P., Thompson, P. M., Szeszko, P., Robinson, D., Dimtcheva, T., et al. (2007). Relationships between IQ and regional cortical gray matter thickness in healthy adults. *Cerebral Cortex*, *17*, 2163–2171.
Pakkenberg, B., & Gundersen, H. J. (1997). Neocortical neuron number in humans: Effect of sex and age. *Journal of Comparative Neurology*, *384*, 312–320.
Park, J., Carp, J., Hebrank, A., Park, D. C., & Polk, T. A. (2010). Neural specificity predicts fluid processing ability in older adults. *Journal of Neuroscience*, *30*, 9253–9259.
Park, D. C., Polk, T. a, Park, R., Minear, M., Savage, A., & Smith, M. R. (2004). Aging reduces neural specialization in ventral visual cortex. *Proceedings of the National Academy of Sciences of the United States of America*, *101*, 13091–13095.
Payer, D., Marshuetz, C., Sutton, B., Hebrank, A., Welsh, R. C., & Park, D. C. (2006). Decreased neural specialization in old adults on a working memory task. *Neuroreport*, *17*, 487–491.
Perfetti, B., Saggino, A., Ferretti, A., Caulo, M., Romani, G. L., & Onofrj, M. (2009). Differential patterns of cortical activation as a function of fluid reasoning complexity. *Human Brain Mapping*, *30*, 497–510.
Peters, A., & Sethares, C. (2002). Aging and the myelinated fibers in prefrontal cortex and corpus callosum of the monkey. *Journal of Comparative Neurology*, *442*, 277–291.
Plate, T. (2003). *Holographic reduced representations.* Stanford: CLSI Publications.
Polsky, A., Mel, B. W., & Schiller, J. (2004). Computational subunits in thin dendrites of pyramidal cells. *Nature Neuroscience*, *7*, 621–627.
Prabhakaran, V., Smith, J. A. L., Desmond, J. E., Glover, G. H., & Gabrieli, J. D. E. (1997). Neural substrates of fluid reasoning: An fMRI study of neocortical activation during performance of the Raven's progressive matrices test. *Cognitive Psychology*, *33*, 43–63.
Rasmussen, D., & Eliasmith, C. (2011). A neural model of rule generation in inductive reasoning. *Topics in Cognitive Science*, *3*, 140–153.
Raven, J. C., Raven, J., & Court, J. (1991). *Manual for Raven's progressive matrices and vocabulary scales.* Oxford: Oxford Psychologists Press.
Raz, N., Lindenberger, U., Rodrigue, K. M., Kennedy, K. M., Head, D., Williamson, A., et al. (2005). Regional brain changes in aging healthy adults: General trends, individual differences and modifiers. *Cerebral Cortex*, *15*, 1676–1689.

Redgrave, P., Prescott, T. J., & Gurney, K. (1999). The basal ganglia: A vertebrate solution to the selection problem? *Neuroscience*, *89*, 1009–1024.
Resnick, S. M., Pham, D. L., Kraut, M. A., Zonderman, A. B., & Davatzikos, C. (2003). Longitudinal magnetic resonance imaging studies of older adults: A shrinking brain. *Journal of Neuroscience*, *23*, 3295–3301.
Royle, N. A., Booth, T., Hernández, M. C. V., Penke, L., Murray, C., Gow, A. J., et al. (2013). Estimated maximal and current brain volume predict cognitive ability in old age. *Neurobiology of Aging*, *34*(12), 2726–2733.
Salthouse, T. (1993). Influence of working memory on adult age differences in matrix reasoning. *British Journal of Psychology*, *84*, 171–199.
Salthouse, T. (1996). The processing-speed theory of adult age differences in cognition. *Psychological Review*, *103*, 403–428.
Salthouse, T. (2009). When does age-related cognitive decline begin? *Neurobiology of Aging*, *30*, 507–514.
Salthouse, T. (2011). Neuroanatomical substrates of age-related cognitive decline. *Psychological Bulletin*, *137*, 753–784.
Seger, C., & Cincotta, C. (2006). Dynamics of frontal, striatal, and hippocampal systems during rule learning. *Cerebral Cortex*, *16*, 1546–1555.
Staff, R. T., Murray, A. D., Deary, I. J., & Whalley, L. (2006). Generality and specificity in cognitive aging: A volumetric brain analysis. *NeuroImage*, *30*, 1433–1440.
Stewart, T. C., Choo, X., & Eliasmith, C. (2010). Dynamic behaviour of a spiking model of action selection in the basal ganglia. In S. Ohlsson, & R. Catrambone (Eds.), *Proceedings of the 32nd annual conference of the Cognitive Science Society* (pp. 235–240). Austin: Cognitive Science Society.
Stewart, T. C., Tang, Y., & Eliasmith, C. (2011). A biologically realistic cleanup memory: Autoassociation in spiking neurons. *Cognitive Systems Research*, *12*, 84–92.
Sullivan, E. V., Adalsteinsson, E., & Pfefferbaum, A. (2006). Selective age-related degradation of anterior callosal fiber bundles quantified in vivo with fiber tracking. *Cerebral Cortex*, *16*, 1030–1039.
Tamez, E., Myerson, J., & Hale, S. (2008). Learning, working memory, and intelligence revisited. *Behavioural Processes*, *78*, 240–245.
Tamez, E., Myerson, J., & Hale, S. (2012). Contributions of associative learning to age and individual differences in fluid intelligence. *Intelligence*, *40*, 518–529.
Thibadeau, R., Just, M., & Carpenter, P. (1982). A model of the time course and content of reading. *Cognitive Science*, *203*, 157–203.
Tucker-Drob, E. M. (2010). Global and domain-specific changes in cognition throughout adulthood. *Developmental Psychology*, *47*, 331–343.
Unsworth, N., & Engle, R. W. (2005). Working memory capacity and fluid abilities: Examining the correlation between Operation Span and Raven. *Intelligence*, *33*, 67–81.
Van De Vijver, F. (1997). Meta-analysis of cross-cultural comparisons of cognitive test performance. *Journal of Cross-Cultural Psychology*, *28*, 678–709.
Vigneau, F., Caissie, A., & Bors, D. (2006). Eye-movement analysis demonstrates strategic influences on intelligence. *Intelligence*, *34*, 261–272.
Wiley, J., Jarosz, A. F., Cushen, P. J., & Colflesh, G. J. H. (2011). New rule use drives the relation between working memory capacity and Raven's advanced progressive matrices. *Journal of Experimental Psychology*, *37*, 256–263.
Zarahn, E., Rakitin, B., Abela, D., Flynn, J., & Stern, Y. (2007). Age-related changes in brain activation during a delayed item recognition task. *Neurobiology of Aging*, *28*, 784–798.