UC Merced
Proceedings of the Annual Meeting of the Cognitive Science Society

Title
A Neural Model of Rule Generation in Inductive Reasoning

Permalink
https://escholarship.org/uc/item/9x09p399

Journal
Proceedings of the Annual Meeting of the Cognitive Science Society, 32(32)

ISSN
1069-7977

Authors
Rasmussen, Daniel
Eliasmith, Chris

Publication Date
2010

Peer reviewed

eScholarship.org Powered by the California Digital Library
University of California

A Neural Model of Rule Generation in Inductive Reasoning

Daniel Rasmussen (drasmuss@uwaterloo.ca)
Chris Eliasmith (celiasmith@uwaterloo.ca)
Centre for Theoretical Neuroscience, University of Waterloo
Waterloo, ON, Canada, N2J 3G1

Abstract

Inductive reasoning is a fundamental and complex aspect of human intelligence. In particular, how do subjects, given a set of particular examples, generate general descriptions of the rules governing that set? We present a biologically plausible method of accomplishing this task, and implement it in a spiking neuron model. We demonstrate the success of this model by applying it to the problem domain of Raven's Progressive Matrices, a widely used tool in the field of intelligence testing. The model is able to generate the rules necessary to correctly solve Raven's items, as well as recreate many of the experimental effects observed in human subjects.

Keywords: inductive reasoning; neural engineering framework; fluid intelligence; Raven's Progressive Matrices; vector symbolic architectures; cognitive modeling

Introduction

Inductive reasoning is the process of using a set of examples to infer a general rule which both describes the relationships shared by those examples and allows us to predict future items in the set. For example, if a person were watching objects in a river or lake and saw a stick, a wooden rowboat, and a telephone pole float past, they might induce the rule that "wooden things float". This rule both describes the relationship which linked those items (being wooden) and allows the person to predict future items which would also float (a wooden bookcase). Given even more examples—some non-wooden floating objects—they might infer the general rule that objects float when they displace a volume of water equal to their weight.

This type of reasoning is fundamental to our ability to make sense of the world, and represents a key facet of human intelligence. It determines our ability to be presented with a novel situation or problem and extract meaning from it. As such, it is a process which has been made central to many tests of general intelligence. One of the most widely used and well respected tools in this field is the Raven's Progressive Matrices (RPM) test (Raven, 1962). In the RPM, subjects are presented with a 3x3 matrix, in which each cell in the matrix contains various geometrical figures with the exception of the final cell which is blank (Figure 1). The subject's task is to determine which one of eight possible answers belongs in the blank cell. They accomplish this by examining the other rows and columns and inducing rules which govern the features in those cells. They can then apply those rules to the last row/column to determine which answer belongs in the blank cell.

Although there has been much experimental and theoretical effort put into understanding the mental processes involved in performing RPM-like tasks, to our knowledge there have been no models of the inductive process of rule generation. In this paper we present a method of rule generation, and implement it in a neural model using simulated spiking neurons. This model can induce the rules necessary to solve Raven's matrices, and also displays many of the most interesting cognitive effects observed in humans: improved accuracy in rule generation over multiple trials, variable performance in repeated trials, and both quantitative and qualitative changes in individual performance.

Background

Raven's Progressive Matrices

There are several variations of the RPM; the Standard and Coloured versions are generally used to test children or adults with cognitive deficits, while the Advanced is used to differentiate average/above-average adults. In our work we focus on the Advanced version.

Figure 1 depicts an example of a simple Raven's-style matrix.[1] The matrix is shown at the top with one blank cell, and the 8 possible candidates for that blank cell are along the bottom. In order to solve this matrix the subject needs to generate three rules: 1) the number of instances of each shape increases by one across the row, 2) the orientation of the shapes within a cell is constant across the row, 3) each cell in a row contains one shape type from the set {square, triangle, circle}. Subjects can then determine which elements belong in the blank cell by applying the rules to the third row (i.e. there should be $2+1=3$ shapes, they should be arranged in the same orientation (vertically), and they should be triangles, since circle and square are already taken). Once they have

[1]For copyright reasons we have created a modified matrix to present here, the model works with the true Raven's matrices.

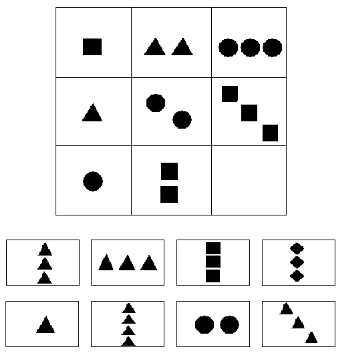
Figure 1: A simple Raven's-style matrix

generated their hypothesis as to what the blank cell should look like, they can check for a match among the 8 possible answers. Not all subjects will explicitly generate these exact rules, and their route to the answer may be more roundabout, but they do need to extract equivalent information if they are to correctly solve the problem.

Despite the test's broad use, the only other computational model for the RPM is that of Carpenter et al. (1990). Their model accurately recreates high-level human data, but does not reflect the flexibility and variability of individual human performance nor take into account neurological data. In addition, Carpenter et al.'s model has no ability to generate new rules; all the rules are pre-programmed. This limitation of their model reflects a general lack of explanation in the literature as to how this inductive process is performed.

The two default assumptions regarding the origin of the rules are that people are either 1) born with, or 2) learn earlier in life, a library of rules. During the RPM, these pre-existing rules are then applied to the current inductive problem. Hunt described this theory as early as 1973, and also pointed out the necessary conclusion of this explanation: if RPM performance is dependent on a library of known rules, then the RPM is testing our crystallized intelligence (our ability to acquire and use knowledge or experience) rather than fluid intelligence (our novel problem solving ability). In other words, the RPM would be a similar task to acquiring a large vocabulary and using it to communicate well. However, this is in direct contradiction to the experimental evidence, which shows the RPM strongly and consistently correlating with other measures of fluid intelligence (Marshalek et al., 1983), and psychometric/neuroimaging practice, which uses the RPM as an index of subjects' fluid reasoning ability (Perfetti et al., 2009; Prabhakaran et al., 1997; Gray et al., 2003). A large amount of work has been informed by the assumption that the RPM measures fluid intelligence, yet the problem raised by Hunt has been largely ignored. Consequently, there is a need for a better explanation of rule induction; by providing a technique to dynamically generate rules, we remove the dependence on a past library, and thereby resolve the problem.

In contrast to the paucity of theoretical results, there has been an abundance of experimental work on the RPM. This has brought to light a number of important aspects of human performance on the test that need to be accounted for by any potential model. First, there are a number of learning effects: subjects improve with practice if given the RPM multiple times (Bors & Vigneau, 2003), and also show learning within the span of a single test (Verguts & De Boeck, 2002). Second, there are both qualitative and quantitative differences in individuals' ability; they exhibit the expected variability in "processing power" (variously attributed to working memory, attention, learning ability, or executive functions), but also consistent differences in high-level problem-solving strategy between low-scoring and high-scoring individuals (Vigneau et al., 2006). Third, a given subject's performance is far from deterministic; given the same test multiple times, subjects will get previously correct answers wrong and vice versa (Bors & Vigneau, 2003). In the Results section we demonstrate how each of these observations is accounted for by our model.

Vector encoding

In order to represent a Raven's matrix in neurons and work on it computationally, we need to translate the visual information into a symbolic form. Vector Symbolic Architectures (VSAs; Gayler, 2003) are one set of proposals for how to construct such representations. VSAs represent information as vectors, and implement mathematical operations to combine those vectors in meaningful ways.

To implement a VSA it is essential to define a binding operation (which ties two vectors together) and a superposition operation (which combines vectors into a set). We use circular convolution for binding, and vector addition for superposition (Plate, 2003). Circular convolution is defined as

$$\begin{aligned} C &= A \otimes B \\ \text{where} & \\ c_j &= \sum_{k=0}^{n-1} a_k b_{j-k \bmod n} \end{aligned} \tag{1}$$

Along with this we employ the idea of a transformation vector T between two vectors A and B, defined as

$$\begin{aligned} A \otimes T &= B \\ \text{or} & \\ T &= A' \otimes B \end{aligned} \tag{2}$$

where A' denotes the approximate inverse of A.

With these elements we can create a vector representation of the information in any Raven's matrix. For example, suppose we wanted to encode the information contained in the third cell of Figure 1. The first step is to define a vocabulary, the elemental vectors which will be used as building blocks. These vectors are randomly generated, and the number of vectors that can be held in a vocabulary and still be distinguishable as unique "words" is determined by the dimensionality of those vectors (the more words in the vocabulary, the higher the dimension of the vectors needed to represent them).

Once the vocabulary has been generated it is possible to encode the structural information in the third cell. A simple method to do this is by using a set of $attribute \otimes value$ pairs: $shape \otimes circle + number \otimes three + colour \otimes black + orientation \otimes horizontal + shading \otimes solid$ and so on, allowing us to encode arbitrary amounts of information. As descriptions become more detailed it is necessary to use more complex encoding; however, ultimately it does not matter to the inductive system how the VSA descriptions are implemented, as long as they encode the necessary information. Thus these descriptions can be made as simple or as complex as desired without impacting the underlying model.

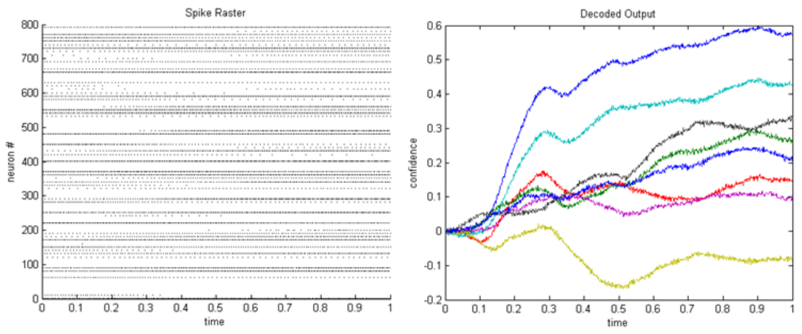

Figure 2: Recordings from the output population of the model, which expresses the similarity between the predicted answer and each of the 8 possible choices. On the left is the spike raster, and on the right is the decoded information from those spikes. The model correctly picks answer number one (the top line).

VSAs have a number of other advantages: vectors are easier to represent in populations of neurons than complex visual information, they are easier to manipulate mathematically, and perhaps most importantly the logical operation of the inductive system is not dependent on the details of the visual system. All that our neural model requires is that the Raven's matrices are represented in some structured vector form; the visual processing which accomplishes this, though a very difficult and interesting problem in itself (see Meo et al. 2007 for an example of the complexities involved), is beyond the scope of the current model. This helps preserve the generality of the inductive system: the techniques presented here will apply to any problem that can be represented in VSAs, not only problems sharing the visual structure of the RPM.

Neural encoding

Having described a method to represent the high-level problem in structured vectors, we now define how to represent those vectors and carry out the VSA operations in networks of simulated spiking neurons. There are several important reasons to consider a neural model. First, by tying the model to the biology we are better able to relate the results of the model to the experimental human data, both at the low level (eg. fMRI or PET) and at the high level (eg. non-deterministic performance and individual differences). Second, our goal is to model human inductive processes, so it is essential to determine whether or not a proposed solution can be realized in a neural implementation. Neuroscience has provided us with an abundance of data from the neural level that we can use to provide constraints on the system. This ensures that the end result is indeed a model of the human inductive system, not a theoretical construct with infinite capacity or power.

We use the techniques of the Neural Engineering Framework (Eliasmith & Anderson, 2003) to represent vectors and carry out the necessary mathematical operations in spiking neurons. To encode a vector $x(t)$ into the spike train of neuron a_i we define

$$a_i(x(t)) = G_i\left[\alpha_i\tilde{\phi}_i x(t) + J_i^{bias}\right] \tag{3}$$

G_i is a function representing the nonlinear neuron characteristics—essentially, how will the neuron spike given the input described within the brackets. In our model we use Leaky Integrate and Fire neurons, but the advantage of this formulation is that any neuron model can be substituted for G_i without changing the overall framework. α_i is a gain on the input, determined by the characteristics of this particular neuron. J_i^{bias} is the background current, modelling the activity in the network which is not a direct input to this neuron. $\tilde{\phi}_i$ represents the neuron's preferred stimulus, that is, which inputs will make it fire more strongly. Broadly speaking, the activity of neuron a_i is a result of its unique response (determined by its preferred stimulus) to the input $x(t)$, passed through a nonlinear neuron model in order to generate spikes.

We can then define the decoding from spike train to vector as

$$\hat{x}(t) = \sum_i h(t) * a_i(x(t))\phi_i \tag{4}$$

where $h(t)$ is a model of the post-synaptic current generated by one spike, $a_i(x(t))$ are the spikes generated by Equation 3, and ϕ_i are the optimal linear decoders. The optimal linear decoders are calculated analytically so as to provide the best linear representation of the original input $x(t)$; they are essentially a weight on the post-synaptic current generated by each neuron (the result of summing the current generated by each spike).

We have defined how to transform a vector into neural activity and how to turn that neural activity back into a vector, but we also need to be able to carry out the VSA operations (binding and superposition) on those representations. One of the primary advantages of the NEF is that we can calculate the synaptic weights for arbitrary transformations analytically, rather than learning them. If we want to calculate a transformation of the form $z = C_1x + C_2y$ (C_1 and C_2 are any matrix), and x and y are represented in the a and b neural populations (we can add or remove these terms as necessary to perform operations on different numbers of variables), respectively, then we describe the activity in the output popula-

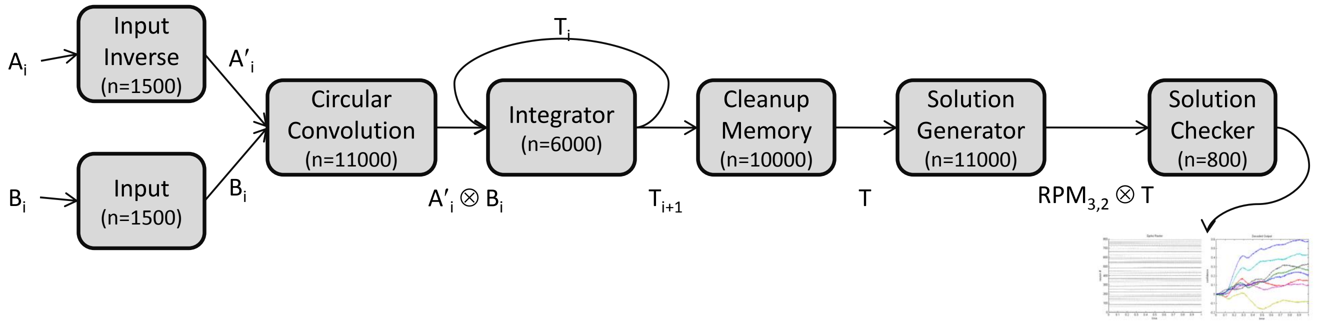

Figure 3: Schematic diagram of the rule generation section with cleanup memory, displaying the approximate number of neurons used in each submodule. The inputs (A_i and B_i) represent two adjacent cells in the matrix. The "Input Inverse" module calculates A_i', while "Input" simply leaves B_i unchanged. The "Circular Convolution" module calculates $A_i' \otimes B_i$ (the rule for that particular pair of cells). "Integrator" is storing the calculated rule so far (based on previous pairs of adjacent cells), which we combine with the current calculation. The output of "Integrator" is the overall rule, which we pass through a cleanup memory, potentially giving us a less noisy version of that rule. Finally, "Solution Generator" generates a prediction of what should be in the blank cell by convolving the second-last cell with our calculated rule, and then "Solution Checker" calculates the similarity between that hypothesis and each of the eight possible answers given in the problem.

tion as

$$c_k(C_1x + C_2y) = G_k\left[\sum_i \omega_{ki}a_i(x) + \sum_j \omega_{kj}b_j(y) + J_k^{bias}\right]$$

where c_k, a_i, and b_j describe the activity of the kth, ith, and jth neuron in their respective populations. The ω are our synaptic weights: $\omega_{ki} = \alpha_k\langle\tilde{\phi}_k C_1 \phi_i^x\rangle_m$ and $\omega_{kj} = \alpha_k\langle\tilde{\phi}_k C_2 \phi_j^y\rangle_m$. Referring back to our descriptions of the variables in Equations 3 and 4, this means that the connection weight between neuron a_i and c_k is determined by the preferred stimulus of c_k, multiplied by the desired transformation and the decoders for a_i. To calculate different transformations all we need to do is modify the C matrices in the weight calculations, allowing us to carry out all the linear computations necessary in this model. For a more detailed description of this process, and a demonstration of implementing the nonlinear circular convolution (Equation 1), see Eliasmith (2005).

The Model and Results

Rule generation

The key to our model is the idea of the transformation vector (Equation 2). Since we have our Raven's matrix items encoded as vectors, we can represent rules as transformations on those vectors. For example, if A is the vector representation of one square, and B is the vector representation of two squares, then the transformation vector $T = A' \otimes B$ will be analogous to the rule "number of squares increases by one". However, we do not just want to calculate individual transformations, we want general rules for the whole matrix. To accomplish this we treat all adjacent pairs of cells as a set of A and B vectors, and extract a general transformation from that set of examples. Neumann (2001) has shown that we can accomplish this by calculating

$$T = \frac{1}{n}\sum_{i=0}^{n} A_i' \otimes B_i$$

In order to perform this operation in neurons (where we cannnot instantly sum over a set of examples) we translate it into the equivalent learning rule, where each pair of A and B vectors is presented sequentially:

$$T_{i+1} = T_i - w_i(T_i - A_i' \otimes B_i)$$

We implement this by combining a neural integrator (to maintain the overall value of T) with a network which calculates the T_i for the current pair of examples. We present the examples in a top-down row-wise fashion, as that is the general scanning strategy employed by humans as revealed by eye-tracking studies (Carpenter et al., 1990; Vigneau et al., 2006). Let us again take Figure 1 as an example, and examine how the model induces one of the rules necessary to solve the matrix: "number of objects increases by one". A_0 is the vector representation of one square, and B_0 is the vector representation of two triangles (we will omit orientation in this example to keep things simple, but it is treated in exactly the same way). The network calculates $T_1 = A_0' \otimes B_0$, which is something like the rule "number of objects increases by one and squares become triangles", and that value is stored in the neural integrator. In the next step A_1 is two triangles and B_1 is three circles, and T_2 is "number of objects increases by one and triangles become circles". However, when T_2 is added to the neural integrator, "number of objects increases by one" is reinforced (since it was already present) while the other information is not. This process continues with the next two rows. Thus we begin with a very noisy rule, but over time relations which are particular to individual A and B pairs are drowned out by the relation which all the pairs have in common: "number of objects increases by one".[2]

Once this process is complete we have the overall T vector, representing a general rule for the problem. Thus we have accomplished our primary goal, to provide an explanation as to how subjects can inductively generate descriptions of the rules governing a set of examples. We use these rules by applying them to the second-last cell of the Raven's matrix $A \otimes T$ giving us B, a vector representing what our rules tell us should be in the blank cell. We then compare this hypothesis to the eight possible answers and take the most similar (determined by the dot product between the two vectors) as our final answer (see Figures 2 and 3).

Cleanup memory

In addition to being able to generate the rules to solve a matrix, the model should improve at this process given practice. We accomplish this by adding a cleanup memory, a system which stores certain values and, when given a noisy version of those values as input, outputs the clean version stored in memory. A cleanup memory can be implemented in neurons by creating a network which contains neural populations tuned to respond only to certain inputs and output the clean version of those values (Stewart et al., 2009). We implement a cleanup memory in this model by storing the past rules the system has induced. The current rule generated by the network, which will be perturbed by neural noise and the details of the particular Raven's matrix, is passed through this cleanup memory, and if the cleanup memory contains a similar rule then that clean version of the rule is output.

The cleanup memory is improved over time by two mechanisms. First, if the cleanup memory receives an input that it does not recognize, it adds that input to its memory so that it will be recognized in the future. Second, if the cleanup memory receives an input that it does recognize, it uses that input to refine the value stored in memory, so that the stored value becomes increasingly accurate. Thus as the system encounters rules it has calculated before it will be able to draw on its past efforts to provide a more accurate output. See Figure 4 for a demonstration of how this improvement in cleanup memory can lead to improved inductive performance.

The cleanup memory not only helps account for observed learning effects, it also bridges the gap between this model of inductive rule generation and theories of a "library" of known rules. In short, we are improving on current theories by explaining where that past knowledge comes from, and why its use is a dynamic, fluid process.

Higher level processes

In addition to the inductive process of rule generation, there are high-level problem solving effects (what we might call the subject's "strategy") which will have a significant impact on performance. For example, how does the subject decide when and where to apply the rule generation system? When there are multiple rules to be found, how does the subject differentiate them, and how do they decide they have found all the rules? How does the subject decide whether their hypothesis is good enough to settle on as a final answer? These are important questions, but they are dependent on the particular problem the subject is solving.

[2]This same process will help eliminate the noise added at the neural level.

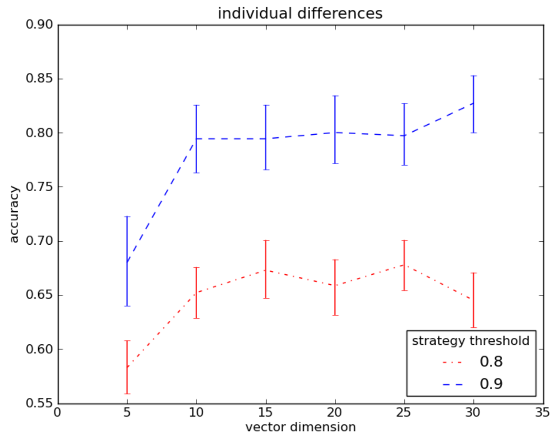

Figure 5: A demonstration of both low-level (vector dimension) and high-level (strategy) influences on accuracy (displaying 95% confidence intervals).

We have implemented such a strategy system for the RPM (although not at the neural level) in order to collect aggregate test results and explore individual differences. Figure 5 shows an example of these results, demonstrating the model's ability to recreate differences caused by both low-level neural processing power and high-level strategy. The low-level variable is the dimensionality of the vectors, higher dimension vectors requiring more neurons to represent. The high-level variable is how willing the model is to decide it has found a correct rule: the lower line represents a subject who has less stringent standards, and is willing to accept rules that may not be completely correct, whereas the top line represents a subject employing a more conservative strategy. These results demonstrate that both low and high level variables have a significant impact on accuracy, and reflect the quantitative and qualitative individual differences observed in human performance. Figure 5 also reveals that although the overall performance trends are clear, there is significant variability (average $\sigma = 0.13$) in any given trial, another parallel of human subjects. There are many such interesting avenues of exploration, however we will not go into the details of the strategy system here; the primary contribution of this research is the general rule-induction system described above, which is not dependent on the higher level framework within which it is used.

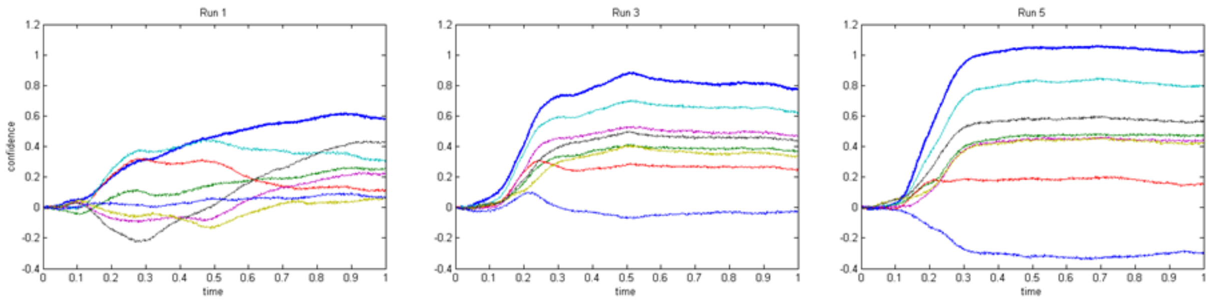

Figure 4: An example of the model's ability to learn over time. The model was presented with a series of matrices that appeared different but required the same underlying rules to solve; as we can see, the model is able to more quickly and definitively pick out the correct answer on later matrices.

Conclusion

We have presented a novel, neurally-based model of inductive rule generation, and we have applied this system to the particular problem of Raven's Progressive Matrices. The success of the system is demonstrated in its ability to correctly find general rules that enable it to solve these matrices, as well as in the model's ability to recreate the interesting effects observed in human subjects, such as learning over time, non-deterministic performance, and both quantitative and qualitative variability of individual differences. These results demonstrate the potential for gaining a deeper understanding of human induction by adopting a neurally plausible approach to modeling cognitive systems.

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.

References

Bors, D., & Vigneau, F. (2003). The effect of practice on Raven's Advanced Progressive Matrices. *Learning and Individual Differences*, *13*, 291-312.

Carpenter, P., Just, M., & Shell, P. (1990). What one intelligence test measures: A theoretical account of the processing in the Raven's Progressive Matrices test. *Psychological Review*, *97*, 404-431.

Eliasmith, C. (2005). Cognition with neurons: A large-scale, biologically realistic model of the Wason task. In B. Bara, L. Barsalou, & M. Bucciarelli (Eds.), *Proceedings of the 27th annual conference of the Cognitive Science Society.* Stresa: Cognitive Science Society.

Eliasmith, C., & Anderson, C. (2003). *Neural engineering: Computation, representation, and dynamics in neurobiological systems*. Cambridge: MIT Press.

Gayler, R. (2003). Vector Symbolic Architectures answer Jackendoff's challenges for cognitive neuroscience. In P. Slezak (Ed.), *ICCS/ASCS international conference on cognitive science* (p. 133-138).

Gray, J., Chabris, C., & Braver, T. (2003). Neural mechanisms of general fluid intelligence. *Nature Neuroscience*, *6*, 316-322.

Hunt, E. (1973). Quote the Raven? Nevermore! In L. Gregg (Ed.), *Knowledge and cognition* (p. 129-157). Potomac: Lawrence Erlbaum Associates.

Marshalek, B., Lohman, D., & Snow, R. (1983). The complexity continuum in the radex and hierarchical models of intelligence. *Intelligence*, *7*, 107-127.

Meo, M., Roberts, M., & Marucci, F. (2007). Element salience as a predictor of item difficulty for Raven's Progressive Matrices. *Intelligence*, *35*, 359-368.

Neumann, J. (2001). *Holistic processing of hierarchical structures in connectionist networks*. Unpublished doctoral dissertation, University of Edinburgh.

Perfetti, B., Saggino, A., Ferretti, A., Caulo, M., Romani, G., & Onofrj, M. (2009). Differential patterns of cortical activation as a function of fluid reasoning complexity. *Human Brain Mapping*, *30*, 497-510.

Plate, T. (2003). *Holographic reduced representations*. Stanford: CLSI Publications.

Prabhakaran, V., Smith, J., Desmond, J., Glover, G., & Gabrieli, J. (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.

Raven, J. (1962). *Advanced progressive matrices (sets I and II)*. London: Lewis.

Stewart, T., Tang, Y., & Eliasmith, C. (2009). A biologically realistic cleanup memory: Autoassociation in spiking neurons. In *9th International Conference on Cognitive Modelling.*

Verguts, T., & De Boeck, P. (2002). The induction of solution rules in Raven's Progressive Matrices test. *European Journal of Cognitive Psychology*, *14*, 521-547.

Vigneau, F., Caissie, A., & Bors, D. (2006). Eye-movement analysis demonstrates strategic influences on intelligence. *Intelligence*, *34*, 261-272.