



Topics in Cognitive Science 3 (2011) 140–153

Copyright © 2011 Cognitive Science Society, Inc. All rights reserved.

ISSN: 1756-8757 print / 1756-8765 online

DOI: 10.1111/j.1756-8765.2010.01127.x

A Neural Model of Rule Generation in Inductive Reasoning

Daniel Rasmussen, Chris Eliasmith

Centre for Theoretical Neuroscience, University of Waterloo

Received 10 September 2010; received in revised form 26 October 2010; accepted 1 November 2010

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 for 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; Raven's Progressive Matrices; Vector Symbolic Architectures; Cognitive modeling; Rule generation; Realistic neural modeling; Fluid intelligence

1. Introduction

Inductive reasoning is the process of using a set of examples to infer a general rule that 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 and saw a stick, a rowboat, and a fencepost float past, he or she 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—he or she might infer the general rule that objects float when they displace a volume of water equal to their weight.

Correspondence should be sent to Daniel Rasmussen, Centre for Theoretical Neuroscience, University of Waterloo, Waterloo, ON, Canada N2J 3G1. E-mail: drasmuss@uwaterloo.ca

This type of reasoning is fundamental to our ability to make sense of the world, and it represents a key facet of human intelligence. It underlies our ability to be presented with a novel situation or problem and extract meaning from it. As such, it is a process that 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 3×3 matrix, in which each cell in the matrix contains various geometrical figures with the exception of the final cell, which is blank (Fig. 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 that 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 cognitive models of the inductive process of rule generation. In this article, 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.

2. Background

2.1. Raven's Progressive Matrices

There are several variations of the RPM; the Standard and Colored versions are generally used to test children or lower performing adults, whereas the Advanced is used to differentiate average/above-average subjects. In our work, we focus on the Advanced version.

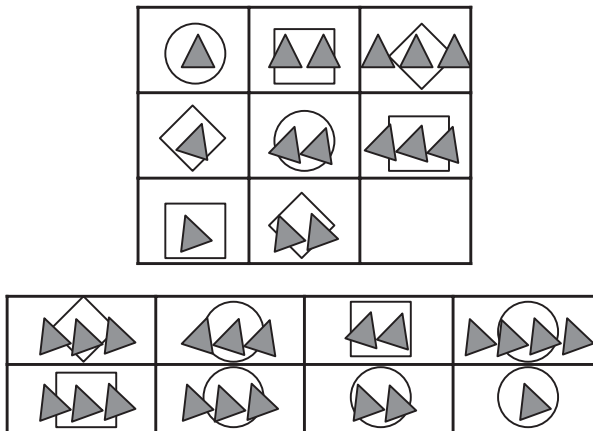


Fig. 1. A simple Raven's-style matrix.

Fig. 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 eight possible answers for that blank cell are given below. In order to solve this matrix, the subject needs to generate three rules: (a) the number of triangles increases by one across the row, (b) the orientation of the triangles is constant across the row, (c) each cell in a row contains one background shape from the set {circle, square, diamond}. Subjects can then determine which element belongs in the blank cell by applying the rules to the third row (i.e., there should be $2 + 1 = 3$ triangles, they should be pointing towards the left, and the background shape should be a circle, since square and diamond are already taken). Once they have generated their hypothesis as to what the blank cell should look like, they can check for a match among the eight 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, there have been few computational models of this task. The model of Carpenter, Just, and Shell (1990) accurately recreates high-level human data (e.g., error rates), but it does not reflect the flexibility and variability of individual human performance nor take into account neurologic data. In addition, Carpenter et al.'s model has no ability to generate new rules; the rules are all specified beforehand by the modelers. This limitation of their model reflects a general lack of explanation in the literature as to how this inductive process is performed. More recently, models have been developed by Lovett, Forbus, and Usher (2010) and McGregor, Kunda, and Goel (2010). The latter employs interesting new techniques based on image processing, but it is not intended to closely reflect human reasoning and is limited to RPM problems that can be solved using visual transformations. The Lovett et al. (2010) model takes an approach more similar to our own and has the advantage of more automated visual processing, but like the Carpenter et al. model it is targeted only at high-level human data and relies on applying rules defined by the modelers.

Previous assumptions regarding the origin of subjects' rules in the RPM are that people are either (a) born with, or (b) learn earlier in life, a library of rules. During the RPM, these preexisting 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, Lohman, & Snow, 1983), and psychometric/neuroimaging practice, which uses the RPM as an index of subjects' fluid reasoning ability (Gray, Chabris, & Braver, 2003; Perfetti et al., 2009; Prabhakaran, Smith, Desmond, Glover, & Gabrieli, 1997). 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, 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) and also consistent differences in high-level problem-solving strategy between low-scoring and high-scoring individuals (Vigneau, Caissie, & Bors, 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, 2003). This is not an exhaustive list, but it represents some of the features that best define human performance. In the Results section, we demonstrate how each of these observations is accounted for by our model.

2.2. 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 necessary 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

$$C = A \otimes B,$$

where

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

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

$$A \otimes T = B \text{ or } T = A' \otimes B, \quad (2)$$

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. The first step is to define a vocabulary, the elemental vectors that will be

used as building blocks. For example, we might use the vector $[0.1, -0.35, 0.17, \dots]$ as the representation for *circle*. 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 a cell. A simple method to do this is by using a set of *attribute* \otimes *value* pairs: *shape* \otimes *circle* + *number* \otimes *three* + *color* \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 overall model.

VSAs have a number of other advantages: They require fewer neural resources to represent than explicit image data, 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 that accomplishes this, although a very difficult and interesting problem in itself (see Meo, Roberts, & Marucci, 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.

2.3. 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 (e.g., fMRI or PET) and at the high level (e.g., nondeterministic performance and individual differences). Second, our goal is to model human inductive processes, so it is essential to determine whether 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. Refer to Fig. 2 throughout this discussion for a visual depiction of the various operations. To encode a vector x into the spike train of neuron a_i we define

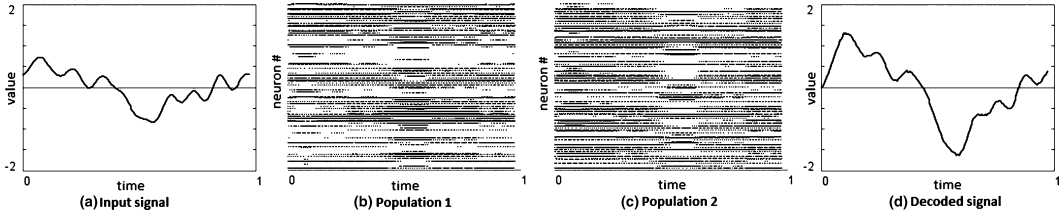


Fig. 2. A demonstration of the elements of the Neural Engineering Framework. (a) to (b) encoding from an input signal to spiking activity in a neural population (Eq. 3). (b) to (c) transforming, in this case doubling, the represented value (Eq. 5). (c) to (d) decoding the spiking activity back into a value (Eq. 4). (Note: (b) and (c) are spike rasters; each row displays one neuron in the population, and each dot in the row represents a spike from that neuron.)

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

G_i as 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, but the advantage of this formulation is that any neuron model can be substituted for G_i without changing the overall framework. α_i , J_i^{bias} , and $\tilde{\phi}_i$ are the parameters of neuron a_i . α_i is a gain on the input; it does not directly play a role in the encoding of information, but rather is used to provide variety in the firing characteristics of the neurons within a population. J_i^{bias} is a constant current arising from intrinsic processes of the cell or background activity in the rest of the nervous system; it plays a similar role to α_i , providing variability in firing characteristics. $\tilde{\phi}_i$ represents the neuron's preferred stimulus, that is, which inputs will make it fire more strongly. This is the most important factor in the neuron's firing, as it is what truly differentiates how a neuron will respond to a given input. In summary, the activity of neuron a_i is a result of its unique response (determined by its preferred stimulus) to the input x , passed through a nonlinear neuron model in order to generate spikes.

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

$$\hat{x} = \sum_i h * a_i(x) \phi_i, \quad (4)$$

where $*$ denotes standard (not circular) convolution. This is modeling the current that will be induced in the postsynaptic cell by the spikes coming out of a_i . $a_i(x)$ are the spikes generated in Eq. 3. h is a model of the postsynaptic current generated by each spike; by convolving that with $a_i(x)$, we get the total current generated by the spikes from a_i . ϕ_i are the optimal linear decoders, which are calculated analytically so as to provide the best linear representation of the original input x ; they are essentially a weight on the postsynaptic current generated by each neuron.

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 population as

$$c_k(C_1x + C_2y) = G_k \left[\sum_i \omega_{ki} a_i(x) + \sum_j \omega_{kj} b_j(y) + J_k^{\text{bias}} \right], \quad (5)$$

where c_k , a_i , and b_j describe the activity of the k th, i th, and j th 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 Eqs. 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 (Eq. 1), see Eliasmith (2005).

3. The model and results

3.1. Rule generation

The key to our model is the idea of the transformation vector (Eq. 2). As 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 cannot 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:

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

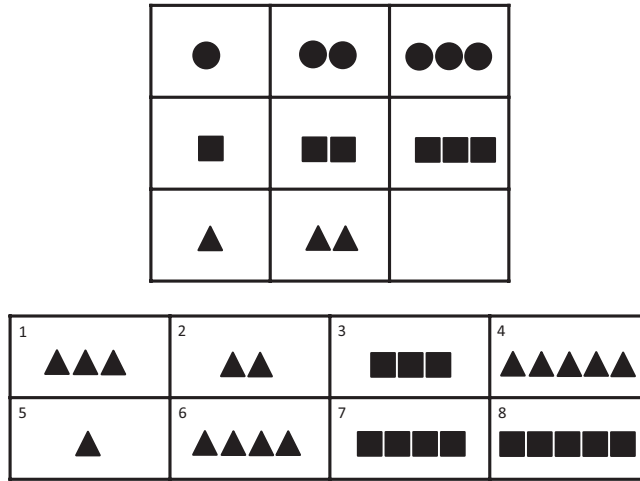


Fig. 3. A simple Raven's-style matrix.

In other words, we calculate an individual transformation for the given pair of cells, and then use the difference between that value and the overall transformation to update the overall transformation for the next pair of cells.

We implement this by combining a neural integrator (to maintain the overall value of T) with a network that calculates the transformation 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 take Fig. 3 as an example and examine how the model induces one of the rules necessary to solve the matrix: “number of objects increases by one.”² A_0 is the vector representation of one circle, and B_0 is the vector representation of two circles. The network calculates $T_1 = A'_0 \otimes B_0$, which is something like the rule “number of circles increases by one,” and that value is stored in the neural integrator. In the next step A_1 is two circles and B_1 is three circles, and the transformation ($A'_1 \otimes B_1$) is again “number of circles increases by one.” However, in the next step, A_2 is one square, B_2 is two squares, and the transformation is “number of squares increases by one.” When this new transformation is added to the neural integrator, “number of objects increases by one” is reinforced (as it is present in all three rules), whereas the specific information (shape) is not. This process continues with the next two rows. Thus, we begin with a specific rule, but over time relations that 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.”³

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 the rules governing a set of examples. We use these rules (T) by applying them to the second-last cell of the Raven's matrix (A) giving us $A \otimes T = B$, where B is a vector representing what our rules tell us should be in the blank

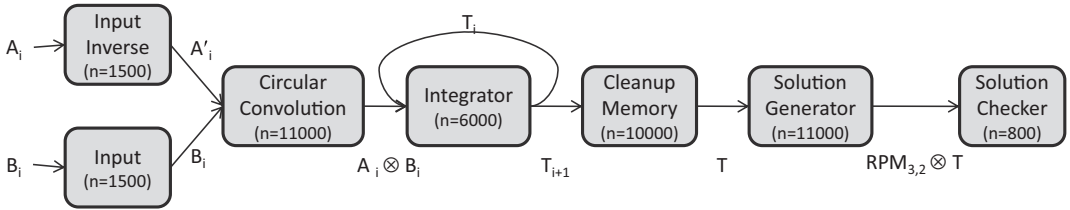


Fig. 4. Schematic diagram of the rule generation section with cleanup memory, displaying the approximate number of neurons used in each submodule. The inputs (A_i and B_i) represent two adjacent cells in the matrix. The “Input Inverse” module calculates A'_i , whereas “Input” simply leaves B_i unchanged. 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 is combined with the current calculation. The output of “Integrator” is the overall rule, which is passed through a cleanup memory, potentially resulting in 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 the calculated rule, and then “Solution Checker” calculates the similarity between that hypothesis and each of the eight possible answers given in the problem.

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 Fig. 4).

The key to induction is that the rules that are generated apply beyond the examples from which they were learned. When examining objects floating in a river, a rule that said “sticks, rowboats, and fenceposts float in rivers” would not be very interesting; an inductive rule (e.g., “wooden things float”) is useful because it applies to new situations—it tells us that a log should float in a lake, although we did not see any logs or lakes when coming up with the rule. We can demonstrate the generality of the rules induced by this system in the same way, by applying them in novel circumstances. The process described above results in a transformation vector T (the rule). Instead of convolving this vector with the second-last cell of the matrix in order to generate a prediction for the blank cell, we can convolve the rule with different vectors and see which answer the system predicts as the next in the sequence. Fig. 5 shows the result of taking the rule generated from the matrix in Fig. 3 (which, when employed in the standard way, predicts the correct answer of three triangles) and applying it instead to the vector for four squares in order to generate a prediction for the blank cell. Note that the vectors for four squares and five squares were not in the examples from which the rule was learned, and yet it still predicts the correct answer. This demonstrates that the rule is not specific to shape (e.g., “number of triangles increases by one”) or number (e.g., “two objects becomes three objects”). The system has correctly extracted a general rule (“number of objects increases by one”) based only on the specific information contained in the matrix.

3.2. 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

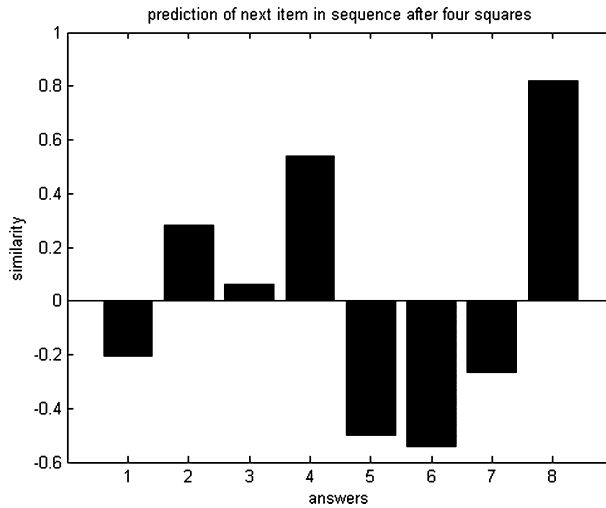


Fig. 5. Result of taking the rule generated from Fig. 3, convolving it with the vector for four squares, and comparing the resulting vector to the eight answers in the matrix. The most similar answer (i.e., the system's prediction of which item comes next in the sequence after four squares) is number eight (five squares).

creating a network that contains neural populations tuned to respond only to certain inputs and output the clean version of those values (Stewart, Tang, & Eliasmith, 2009). We incorporate 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 (see Fig. 4).

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 more accurate output. See Fig. 6 for a demonstration of how this improvement in cleanup memory can lead to improved inductive performance.

The cleanup memory is useful in that it improves the accuracy of the system and accounts for observed learning effects, but it also serves an important theoretical purpose: It bridges the gap between this model of dynamic rule generation and previous theories of a library of known rules. Rather than contradicting previous theories, we are improving on them by explaining where that past knowledge comes from. We now have an explanation as to why the use of that knowledge is a dynamic, fluid process rather than crystallized. The important aspect of a cleanup memory is that it depends upon its input. The cleanup memory can be used to improve the accuracy of rules, but it cannot generate them on demand; the subject needs to first generate an answer that is accurate enough to be recognized. Thus, subjects can still benefit from their past knowledge of rules, but the critical aspect of performance will

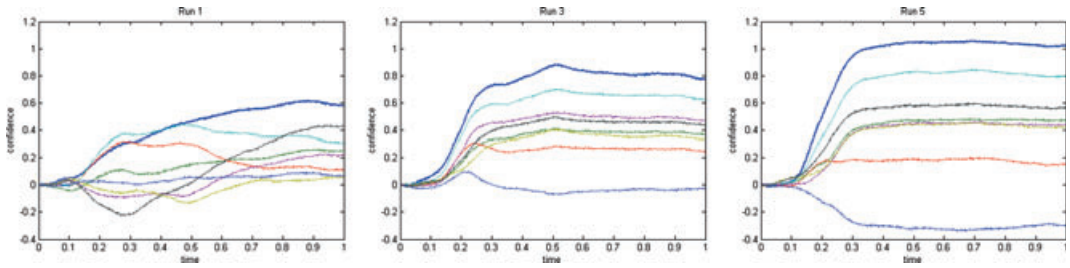


Fig. 6. An example of the model's ability to learn over time. The model was presented with a series of matrices that appeared different but required the same underlying rules to solve; as we can see, the model is able to more quickly and definitively pick out the correct answer on later matrices (the eight lines in each graph represent the system's confidence in the eight possible answers).

be their fluid rule generation ability. This resolves Hunt's dilemma and means that the reliance of previous models on a library of known rules does not render their insights useless; instead, they can simply be reinterpreted from this new, biologically plausible perspective.

3.3. 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") that 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 does the subject decide he or she has found all the rules? How does the subject decide whether his or her hypothesis is good enough to settle on as a final answer? In summary, what does the subject do with the rules once they have been generated? These are important questions, but they are dependent on the particular problem the subject is solving.

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.⁴ Fig. 7 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 variables parallel the common types of explanations for individual differences in human subjects: on one hand, neurophysiologic differences such as gray matter density (Gray et al., 2003), and on the other hand, cognitive, strategic differences (Vigneau et al., 2006). These results demonstrate how the model can be used to investigate how these factors interact to give rise to the full spectrum of individual differences.

Fig. 7 also reveals that although the overall performance trends are clear there is significant variability (average $\sigma = 0.13$) in any given trial. In other words, the same model run

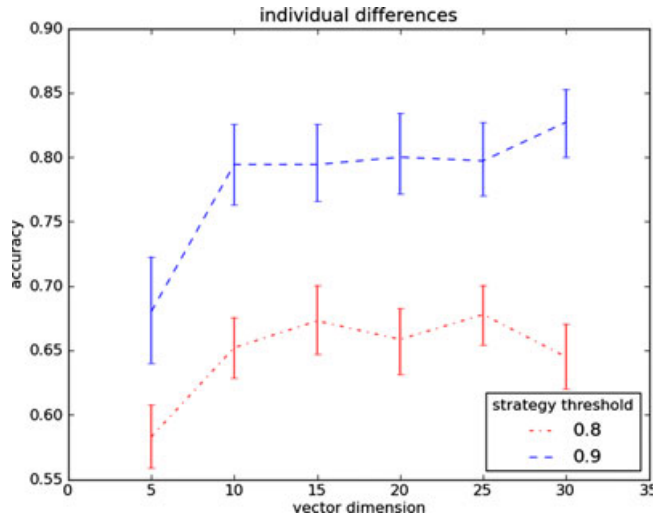


Fig. 7. A demonstration of both low-level (vector dimension/neuron number) and high-level (strategy) influences on accuracy.

repeatedly will get different problems right or wrong, but on an average its performance will be stable. Unlike previous, deterministic models, this is an accurate reflection of the observed patterns of performance in human subjects (Bors, 2003). There are many such interesting avenues of exploration, but 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.

4. 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, nondeterministic performance, and both quantitative and qualitative variability in 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.

Notes

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

2. Note that Fig. 3 differs primarily from Fig. 1 in that the rule involving items chosen from a set has been removed. The model can generate these kinds of rules, but we have not included that description in the current discussion for purposes of brevity.
3. This same process will help eliminate the noise added at the neural level.
4. The strategy system has three main responsibilities: automating the input to the neural modules, evaluating the success of the rules returned by the neural modules, and selecting an overall response when the neural modules find multiple rules (as in Fig. 1). In the current system these are simply programmed solutions, but the model presented in Stewart, Choo, and Eliasmith (2010) is an interesting demonstration of how such processes could be implemented in a realistic neural model.

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. (2003). The effect of practice on Raven's Advanced Progressive Matrices. *Learning and Individual Differences*, 13(4), 291–312.
- 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(3), 404–431.
- Eliasmith, C. (2005). A unified approach to building and controlling spiking attractor networks. *Neural Computation*, 17(6), 1276–1314.
- Eliasmith, C., & Anderson, C. (2003). *Neural engineering: Computation, representation, and dynamics in neurobiological systems*. Cambridge, MA: 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* (pp. 133–138). Sydney: University of New South Wales.
- Gray, J. R., Chabris, C. F., & Braver, T. S. (2003). Neural mechanisms of general fluid intelligence. *Nature Neuroscience*, 6(3), 316–322.
- Hunt, E. (1973). Quote the Raven? Nevermore! In L. Gregg (Ed.), *Knowledge and cognition* (pp. 129–157). Potomac, NJ: Lawrence Erlbaum Associates.
- 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, TX: Cognitive Science Society.
- Marshalek, B., Lohman, D., & Snow, R. (1983). The complexity continuum in the radex and hierarchical models of intelligence. *Intelligence*, 7(2), 107–127.
- McGregor, K., Kunda, M., & Goel, A. (2010). A fractal analogy approach to the Raven's test of intelligence. In *AAAI workshops at the 24th AAAI conference on Artificial Intelligence* (pp. 69–75). Atlanta: Association for the Advancement of Artificial Intelligence.
- Meo, M., Roberts, M., & Marucci, F. (2007). Element salience as a predictor of item difficulty for Raven's Progressive Matrices. *Intelligence*, 35(4), 359–368.

- Neumann, J. (2001). *Holistic processing of hierarchical structures in connectionist networks*, Unpublished doctoral thesis, University of Edinburgh.
- Perfetti, B., Saggino, A., Ferretti, A., Caulo, M., Romani, G. L., & Onofri, M. (2009). Differential patterns of cortical activation as a function of fluid reasoning complexity. *Human Brain Mapping*, 30(2), 497–510.
- Plate, T. (2003). *Holographic reduced representations*. Stanford, CA: CLSI Publications.
- Prabhakaran, V., Smith, J., Desmond, J., Glover, G., & 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.
- Raven, J. (1962). *Advanced progressive matrices (Sets I and II)*. London: Lewis.
- Stewart, T. C., Tang, Y., & Eliasmith, C. (2009). A biologically realistic cleanup memory: Autoassociation in spiking neurons. In A. Howes, D. Peebles, & R. Cooper (Eds.), *9th international conference on cognitive modelling* (pp. 128–133). Manchester, England: ICCM2009.
- 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.
- Verguts, T., & De Boeck, P. (2002). The induction of solution rules in Ravens 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(3), 261–272.