A neural model of rule finding in Raven's Progressive Matrices

Daniel Rasmussen

1 Introduction

Raven's Progressive Matrices is one of the most widely used tests of general intelligence. It has been found to be both domain independent and highly correlated with other measures of intellectual ability [Marshalek et al., 1983]. In the RPM subjects are presented with a 3x3 matrix, where each cell—except for the blank cell in the bottom right—contains multiple features. The task is to determine, given eight possibilities, which answer belongs in the blank cell. Subjects accomplish this by finding the rules that govern the features in each row or column. Once these rules have been found, they can be applied to the last row/column to determine which features will complete the rule in the blank cell. This task requires a complex array of cognitive abilities, and so represents an interesting challenge for understanding human intelligence.

There are many theories as to how subjects go about solving these matrices. Over the years researchers have focused on topics including the different types of rules (Carpenter et al. [1990], DeShon et al. [1995]), error types (Babcock [2002]), the importance of early visual processing (Meo et al. [2007]), working memory (Kyllonen and Christal [1990]), and executive functions (Unsworth and Engle [2005]). There are two significant tasks which stand out as necessary to flesh out this research. The first is developing a working model that combines these theories to recreate human performance. This is the proverbial proof in the pudding; if our theories about how subjects solve the RPM are correct, then we should be able to use those theories to solve the RPM. The only work in this area has been done by Carpenter et al. [1990], who created a model that could solve several types of Raven's matrices. However, their model is now several years old, and does not take into account much of the neuroimaging data (particularly fMRI) which is now available. In addition, the focus of the model was on recreating high-level human performance (ex. error rates) rather than realistically describing human problem solving techniques at a neural or brain system level.

The second gap in the RPM literature is an explanation of how subjects create new rules. In the Carpenter et al. model the rules were preprogrammed; the system had no ability to develop novel solutions. One possible theory is given by Verguts et al. [2000], who suggested that rule finding was a type of

statistical sampling of the space of possible rules. However, it is unlikely that problem solving is nothing more than blind sampling, relying on luck to arrive at the correct answer. We still have no explanation of how this sampling is guided. Indeed, the later research of Verguts and De Boeck [2002] on learning in the RPM suggests that this sampling explanation is more suited to describing our search over known rules, rather than how we come up with novel solutions. What is missing is that crucial intermediate step; we have theories on how visual information is collected in RPM situations, and given a set of rules we have theories on how they are used to solve the matrix, but we are lacking in theories that explain how we make that move from visual information to rule.

In this paper we hope to work towards alleviating both of these concerns. We will provide a mechanistic, neurally plausible model that can find rules based on matrix data and solve simple RPM-like problems. It can by no means be said that this model can recreate human results on the RPM. What we have is a preliminary model of a specific aspect of matrix solving. However, hopefully this model will serve as a stepping stone to future work, and an indication that this is not a fruitless path to explore.

2 System Description

Both behavioural (DeShon et al. [1995], Meo et al. [2007]) and neurological (Haier et al. [1988], Prabhakaran et al. [1997]) data suggest that there are two distinct components to RPM solving. First there is low-level visuospatial processing that is applied directly to the matrix data. Rules resulting from these processes operate only on the visual features of the matrix (ex. combination, movement, or rotation of features). For more difficult problems it is necessary to abstract from the visual data and form logical propositions. Rules resulting from these processes operate on meta-information, such as the fact that all of the shapes in a row are different, rather than the visual data itself (the shapes). In our model we only address the former—low level visual processing. We would like to grow our model to incorporate both, but for now we will leave abstract reasoning aside.

2.1 Neural data

Even restricting ourselves to this domain, the task is so general and widespread that it is difficult to identify specific brain areas responsible. Neuroimaging data from subjects taking the RPM, even when we attempt to isolate visuospatial processing, shows bilateral (although predominantly right-hemisphere) activation in all four lobes. Therefore we can make guesses as to where the processing in this model might be occurring, but we should treat these descriptions as predictions of the model.

Our neuroimaging data for the RPM comes from two sources: Haier et al. [1988] and Prabhakaran et al. [1997]. Haier et al.'s study used PET, so cannot give us detailed activation areas. Broadly speaking, what they found was that

visual processing activated more right hemisphere areas than left hemisphere (the opposite was true for analytic processing), and more posterior areas than anterior.

Our best data comes from Prabhakaran et al.'s fMRI study. When they isolated for visuospatial processing they found primary activation in several of the areas associated with nonverbal working memory, in particular right middle frontal gyrus and bilateral inferior parietal regions. They also found activation in bilateral superior parietal areas associated with the direction of visuospatial attention. Our hypothesis is that these areas are involved in extracting and representing the data from the matrix, which will then become input into our model. Prabhakaran et al. also found activation in the right temporal lobe, in particular the middle and inferior gyri. These areas have been associated with mental rotation, and Prabhakaran et al. suggest that activation "may reflect mental transformations involved in generating a candidate answer for the missing pattern". These are precisely the processes captured in our model. These areas are also situated in the ventral stream of visual processing, which fits with our belief that these visuospatial elements of RPM solving represent low-level, largely automatic visual processes.

2.2 Holographic Reduced Representations

Our model employs HRRs to represent information in a structured manner. A good discussion of HRRs can be found in Plate [2003], which we will only describe in brief here.

HRRs encode information in vectors, and in the neural case we take these vectors to be represented in the firing rates of neural populations (as described in Eliasmith and Anderson [2003]). We will require three operations to create structured representations: circular convolution, superposition, and approximate inverse. Circular convolution is defined as

$$C = A \otimes B$$

where

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

Superposition is simply vector addition, and for the approximate inverse we will use involution:

$$a_i^{-1} = a_{(-i \bmod n)}$$

We can also define the transformation T between two vectors A and B, such that $A \otimes T = B$. All we need to do to find T is rearrange the equation, so $T = A^{-1} \otimes B$. This suffices to find a single transformation, but often we will want to find a general transformation between two classes. Neumann [2001] has described how we can find these T vectors given a series of example A and B

vectors. There are several variations, but essentially it amounts to taking the average of the previous calculation:

$$T = \frac{1}{n} \sum_{i=0}^{n} A_i^{-1} \otimes B_i$$

The key to our model is the idea that if we encode a matrix in HRR form, then finding the rule that governs a row is analogous to finding a vector T that describes how the HRR description of the first cell is transformed into the description of the second cell, and the second cell into the third cell.

2.3 Neural computations

One of the great advantages of the Neural Engineering Framework is that we can calculate the synaptic weights for arbitrary linear transformations analytically, rather than learning them. Eliasmith and Anderson [2003] describe how we can do this. We will not repeat the entire derivation here, but the end result is that if we want to calculate a transformation of the form $z = C_1 x + C_2 y$ (C_1 and C_2 are any matrix), and x, y, and z are stored in the a, b, and c neural populations, respectively, then

$$c_k(C_1 x + C_2 y) = G_k \left[\sum_i \omega_{ki} a_i(x) + \sum_j \omega_{kj} b_j(y) + J_k^{bias} \right]$$

where G_k is a function representing the nonlinear neuron characteristics (in our model we use Leaky Integrate and Fire or LIF neurons). c_k, a_i, and b_j are the firing rates of the kth, ith, and jth neuron in their respective populations, and J_k^{bias} is the background current. The ω are our synaptic weights: $\omega_{ki} = \alpha_k \langle \tilde{\phi_k} C_1 \phi_i^x \rangle_m$ and $\omega_{kj} = \alpha_k \langle \tilde{\phi_k} C_2 \phi_j^y \rangle_m$. If we want to do a linear transformation on only one variable, we simply omit the b population from the above calculation. To calculate different transformations all we need to do is modify the C matrices in the weight calculations; therefore rather than repeating the entire formula in the future we will simply describe the C matrix.

Our model only requires a few relatively simple processing elements, which we will now briefly describe.

2.3.1 Element-wise product

Taking the element-wise product of two vectors requires three steps. First, we separate each component of both the vectors into their own scalar representations. To transform the d dimensional vector A into d 1 dimensional a_i populations, we form d populations and link the ith population to the input population via a dx1 C matrix where $C[i] = 1$ and $C[j \neq i] = 0$. Second, we multiply the appropriate scalar values together (i.e. the first component of A with the first component of B, and so on). For a description of multiplication in the NEF see Eliasmith and Anderson [2003]. Third, we combine the resulting

scalar values back into a single vector. To do this we form a projection from the d 1 dimensional populations into a single d dimensional population, where each projection has a C matrix that is equal to the transpose of the one in the first step. In reality we can combine the first and second steps, they are separated here only for clarity.

2.3.2 Circular convolution

Circular convolution is made easier by noting that it is equivalent to an element-wise product in the frequency domain. Thus all we need to do is define how to perform an FFT and IFFT, and we can combine that with our previous description to achieve circular convolution. Since the FFT is a linear transformation, all we need to do is set C to be the dxd DFT matrix W where $W_{ij} = \cos(\frac{-2\pi \times (ij)}{d}) + \sin(\frac{-2\pi \times (ij)}{d})i$. The inverse DFT matrix is simply the complex conjugate of this.

2.3.3 Transformation calculation

Next we need to define how to calculate a transformation vector. Fortunately, finding T such that $A \otimes T = B$ is as easy as calculating the circular convolution of A^{-1} and B. Thus all we need to do is define how to calculate A^{-1}, and we can combine that with the previous description to calculate T. Recall that we are using the approximate inverse, or involution, where $a_i^{-1} = a_{(-i \bmod n)}$. Therefore to create the C matrix we simply rearrange the dxd identity matrix in the same way (i.e. row i becomes row $-i \bmod d$).

2.3.4 Similarity testing

The final calculation is determining the similarity between two HRRs, defined as the dot product of the two vectors. However, the dot product is nothing more than the sum over the dimensions of the element-wise product, so all we need to do is sum the output of the element-wise product component.

3 Design specification

Our work so far in this model is mainly theoretical. We have defined neural populations and the synaptic weights connecting these populations, but the implementation details will be a subject of further study. By experimenting with different parameter settings and examining how they affect the performance of the system we hope to be able to make predictions about the corresponding neural properties in the brain. There are several properties we will be focusing on:

- Encoding vectors: should these be randomly chosen from the unit circle, or aimed in specific directions? Since calculating the element-wise product, which involves several multiplications, is central to our model, it may

be desirable to do this very accurately. The accuracy of multiplication can be increased by choosing our encoding vectors to equally encode each direction (a "45 degree angle" in higher dimensions).

- Response range: we would expect that the standard response range of ± 1, corresponding to a normalized HRR vector length of 1, will be appropriate for our system. However, circular convolution and superposition do not preserve vector length, so vectors will tend to grow longer. We could renormalize the vector after each computation (although normalization is an expensive operation), or we could expand the response range of the neurons.

- Vector dimension: One of the most interesting variables will be the dimension of the HRR vectors we use in our model. Adding more dimensions allows greater accuracy of encoding/decoding, and thereby allows us to have a larger vocabulary of vector words (since we can distinguish them more accurately). In the current model, with a very simple vocabulary, 10 dimensions seems to be the minimum necessary. However, this will certainly increase as we attempt more difficult matrices requiring more complex descriptions.

- Population size: Vector dimension is a critical issue because it affects the size of the population required to represent that vector. Population size must scale approximately linearly with vector dimension in order to maintain equivalent accuracy. This affects our model practically by determining both the memory size and run time of our model. However, it also determines our prediction of area or number of neurons required to implement our model in the brain; if we were to find that our model necessitated such high dimensional vectors that a significant portion of the neurons in the temporal lobe were required to represent them, then that would force us to rethink the plausibility of our model.

All of these factors are things we hope to examine as our model progresses and we move into a more realistic representation in terms of spiking neurons. If we are correct that these operations are occurring in the right middle/inferior temporal gyri, then after determining appropriate settings for these values we would expect to find matching data from actual brain studies in that area.

4 Implementation

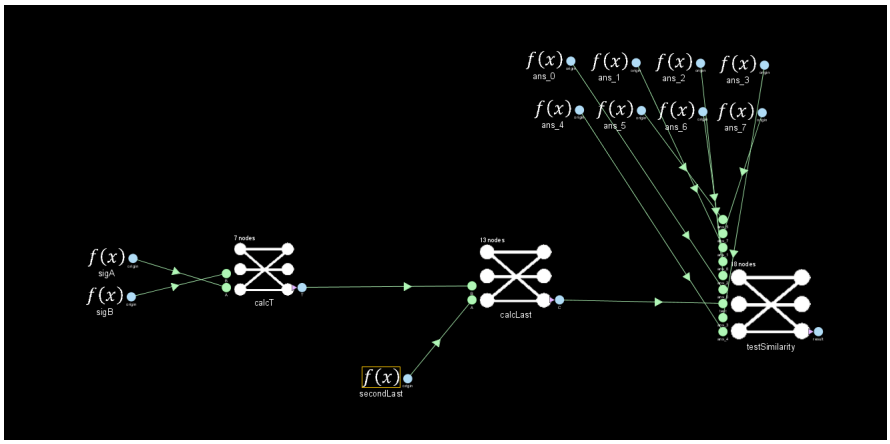
Our model is implemented in Nengo, a standardized Java implementation of the Neural Engineering Framework. Nengo provides a Python scripting interface, which allows us to describe models in a high level language and then see them implemented in a graphical interface [Stewart et al., 2009]. We chose to implement our model in Nengo for a number of reasons. First, it saves on both development time and error rates by utilizing code that has already been empirically tested on a number of models. Second, the GUI provides us with

an intuitive, high-level description of our model, and allows us to make modifications on the fly without returning back to the code. Third, by using the standard Java interface we make it easy to integrate our model with any other system developed in Nengo, or to share our model with other NEF researchers; this facilitates the development of integrated models of various brain systems.

4.1 Model description

We will begin with a high-level description of our model, and then clarify how each component operates. The mathematical description of these components has already been given in the system description, so here we will try to provide a more intuitive understanding.

4.1.1 Top level



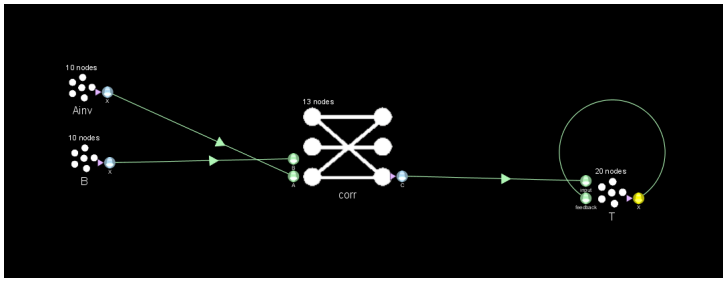
At the top level we see three main components, which we will describe from left to right. First is calcT, which calculates an average T given a series of A and B examples (recall that T is the vector such that $A \otimes T = B$). sigA and sigB are piecewise functions that present the pairs of A and B vectors to the system for 1s each. These A and B vectors are chosen from the pairwise comparisons of neighbouring cells in a row of the matrix (i.e. we present cell[1][1] and cell[1][2], then cell[1][2] and cell[1][3], then cell[2][1] and cell[2][2], and so on). Eye-tracking studies [Carpenter et al., 1990] suggest that this is in fact the method by which people build up rules. The resulting output is the average transformation over all these examples.

Next we take the output from calcT and convolve it with the second last cell (cell[3][2]). cell[3][2] $\otimes$ T should give us cell[3][3]; this is the output from calcLast.

At this point our system has generated a hypothesis of what the last cell should be. We then compare this hypothesis to the 8 possible answers given in the problem. testSimilarity calculates how similar the hypothesis is to each of the answers, and outputs an 8 dimensional vector. The component with the

highest value is our model's best guess at the correct answer. This information then leaves our system to be processed elsewhere, such as translating it into motor movements that will point to the correct answer.

4.1.2 calcT

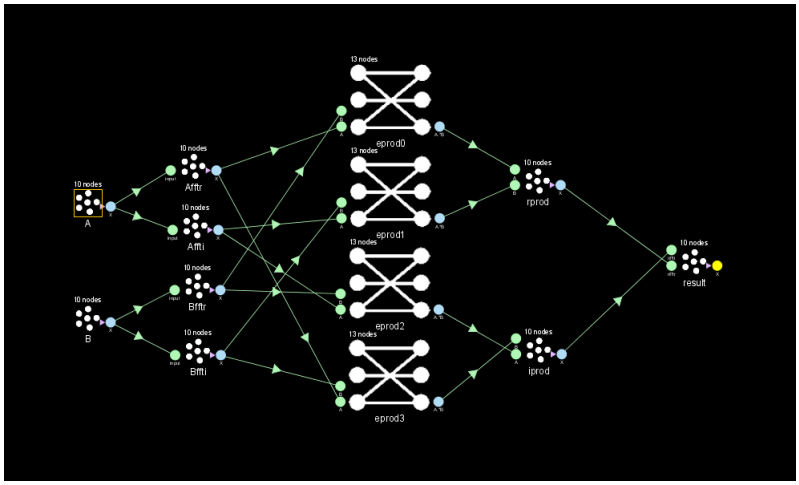


We can now take a closer look at calcT. The two input signals (sigA and sigB) go into the input populations Ainv and B, respectively. Ainv calculates the approximate inverse of A (as described in the system description) while B simply leaves the value unchanged.

Corr then calculates the circular convolution of the output of these two populations (termed corr because the circular convolution of A^{-1} and B is equal to the circular correlation of A and B), which is equal to the T vector for that pair of A and B vectors. We will look closer at calculating the circular convolution in calcLast.

Meanwhile, our previous best estimate of the transformation is being maintained in the population T. T is a modified integrator where the input has a weight of τ^{PSC} and the feedback loop has a weight of $1 - \tau^{PSC}$ (see Eliasmith and Anderson [2003] for a description of building integrators in the NEF). This will cause the representation in T to slowly drift towards whatever value is currently coming out of corr. Since what is coming out of corr is the sequence of T vectors for the various pairs of As and Bs, the effect of this will be to calculate a moving average of T.

4.1.3 calcLast



calcLast is simply the circular convolution of its two inputs, so we will take this opportunity to take a closer look at that operation. The A and B populations represent the two vectors we want to convolve, and the first thing we want to do is take the FFT of each of these vectors. Taking the FFT results in a complex number, which has both a real and imaginary component. We could represent these in one population, but it simplifies the rest of the model if we split them into two . This is what is occurring in the Afftr and Affti populations; they are the real and imaginary components, respectively, of the FFT of the A vector (and B has matching counterparts).

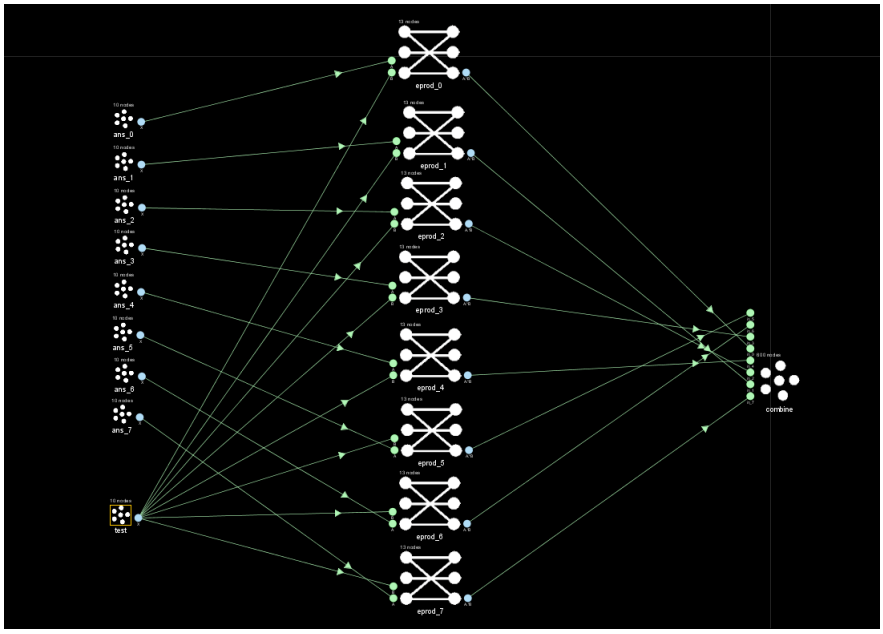
Now we need to take the element-wise product. Unfortunately, it is not as simple as taking the element-wise product of the real and imaginary components, due to the definition of multiplicaton of complex numbers. If we have two complex numbers $X = A + Bi$ and $Y = C + Di$, then $X \times Y = AC + ADi + BiC + BiDi$ (i.e. as if we are multiplying two polynomials). This rearranges to $(AC - BD) + (AD + BC)i$. Thus we have four element-wise products to calculate: AC, BD, AD, and BC. In our model A = Afftr, B = Affti, C = Bfftr, and D = Bffti; thus AC, BD, AD, and BC correspond to eprod 0, 1, 2, and 3. We then need to calculate $AC - BD$ and $AD + BC$ to get the real and imaginary components; this is what is happening in rprod and iprod.

Now we have successfully calculated the element-wise product of the FFT of our two vectors, and we need to take the IFFT. We do this in the same way as we calculated the FFT of A and B, except applying the inverse DFT matrix to rprod and iprod. Theoretically this would give us four results again, but we can simplify the calculation because we know that the imaginary component of our answer will be zero. We can ignore calculations that would result in a nonzero imaginary component, so all we need to do is multiply rprod by the real part of the inverse W matrix (AC), and iprod by the imaginary part of the inverse W matrix (BD). The imaginary components of iprod and the inverse W matrix cancel to give us negative one, so then we simply subtract BD from AC to find

our final answer.

Note that we could be more efficient by taking advantage of the fact that when we take the FFT of a real number the result will be symmetrical, thus we could get away with only representing half the vector. We will implement this improvement as we move into incorporating spiking neurons, since it will provide us with useful computational savings and improved accuracy.

4.1.4 testSimilarity



testSimilarity looks complex, but it is actually the simplest of the components. It is just doing the same thing 8 times—calculating the dot product of our hypothesis ("test") and the possible solutions. Each of the eprod networks calculates the element-wise product of the hypothesis and one of the answers. These products are then summed across the dimensions (giving us the dot product) and combined into an 8 dimensional vector rather than 8 separate values.

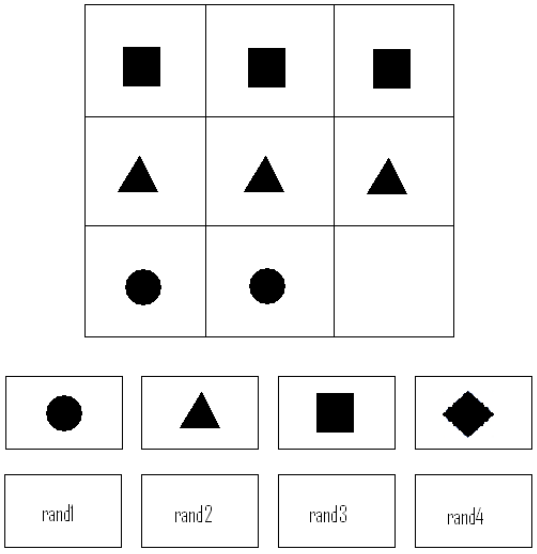
4.2 Results

We have only tried our model on some fairly simple matrices. These matrices are analogous to what we would find on early problems of the Raven's Standard Progressive Matrices, as opposed to the Advanced Progressive Matrices that are usually administered to subjects of average or above average intelligence. However, our model does solve these matrices, using only the simple, neurologically plausible components outlined above.

Note that all of these simulations are being run in Nengo's direct mode; that is, network properties are being simulated but not realistic neurons. As

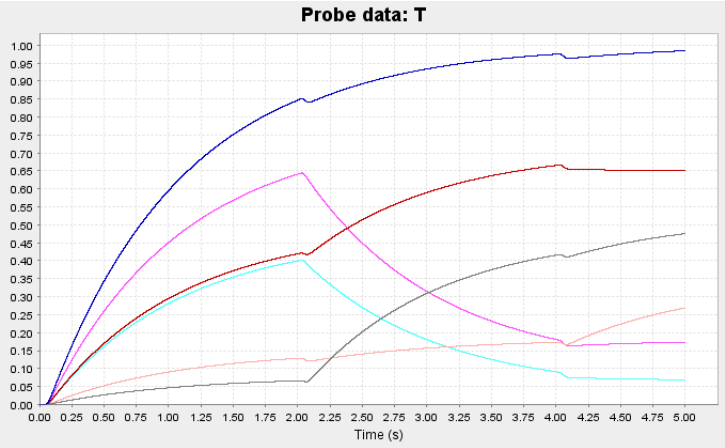
we mentioned in the design specification section, incorporating realistic neural properties is the next step we are pursuing.

4.2.1 Matrix 1

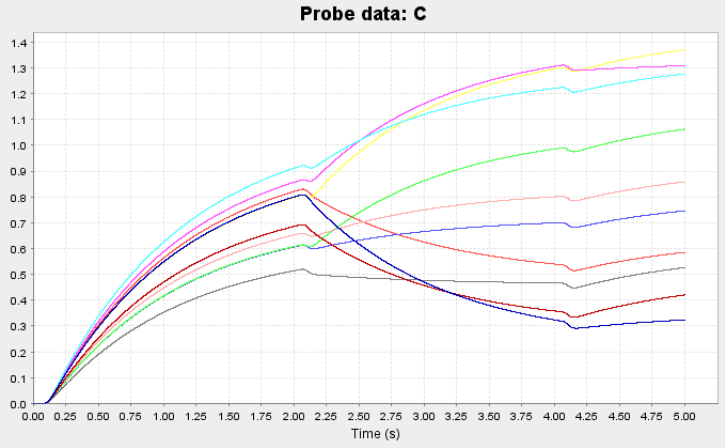


This matrix follows the simple rule that the shape remains constant across the row. The question is whether the system will correctly learn the general rule that the third cell should be the same as the previous cell, rather than that the third cell should be a square or a triangle.

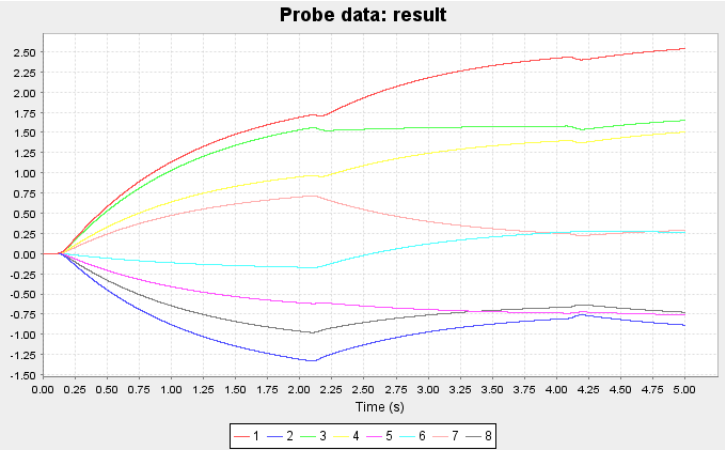
To encode this matrix we will use a very simple vocabulary. For all of these tests our words will be made up of 10 dimensional unit vectors chosen from a normal distribution. For this matrix we will require five of these words: shape, square, triangle, circle and diamond. We will encode cell[1][1] as shape $\otimes$ square, cell[2][1] as shape $\otimes$ triangle, and so on. Our possible answers are given along the bottom, and encoded in the same manner. They are intentionally chosen to challenge the system; if it was going to make an error then it would likely pick triangle or square, so both those options are present. We also add in a shape not found in the matrix, diamond, and four random vectors for comparison.



This figure shows the estimate of T, the transformation vector, over time. We can see that the transformation jumps whenever a novel pair of input vectors is presented (at 2s and 4s, since the first two and second two pairs are identical) but progresses in a relatively constant direction overall.

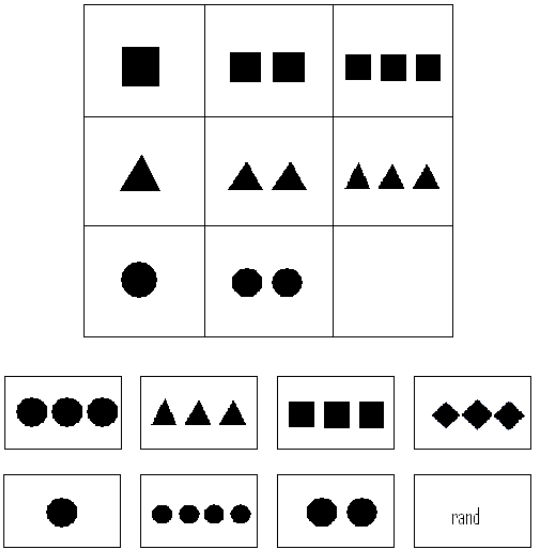


This figure shows the system's estimate of the element in the blank cell over time. Since the vector representing the second last cell is constant, this figure simply reflects the changes in T shown in the previous figure.



This figure shows the system's estimate of which of the possible answers are the best fit for the blank cell. As we can see, the system correctly decides that the correct answer is number 1. We will analyze these results in more detail for the more complicated matrix.

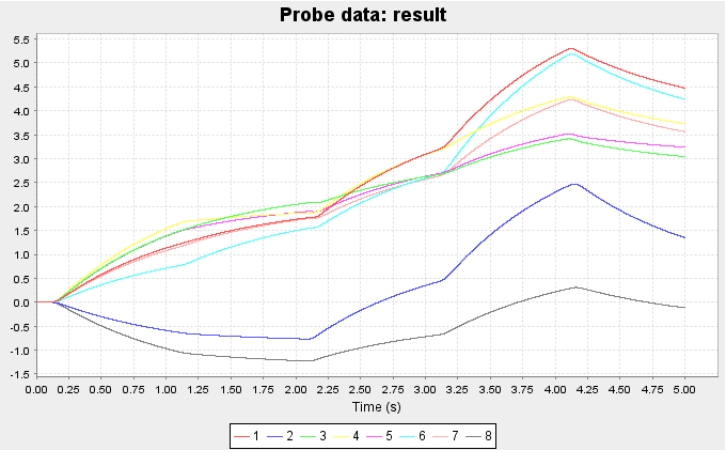
4.2.2 Matrix 2



This matrix is more complex, because in addition to shape we have added the concept of number. Therefore in order to represent it we will need to add additional words to our vocabulary. We will add the words "number", "one", and "plusone". This will allow us to represent arbirarily large numbers by combining one and plusone; two = one $\otimes$ plusone, three = two $\otimes$ plusone ((one $\otimes$ plusone) $\otimes$ plusone), and so on. We then encode the matrix as cell[1][1] =

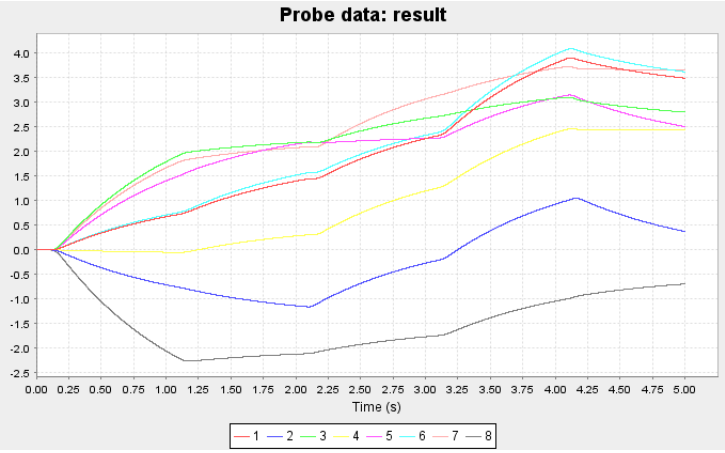
shape ⊗ square + number ⊗ one, cell[1][2] = shape ⊗ square + number ⊗ two, etc.

Now our system needs to learn both that the shape remains constant and that the number increases by one each time. We will only show the results of testSimilarity, since the results of calcT and calcLast are not particularly meaningful and look essentially the same as in the previous case.

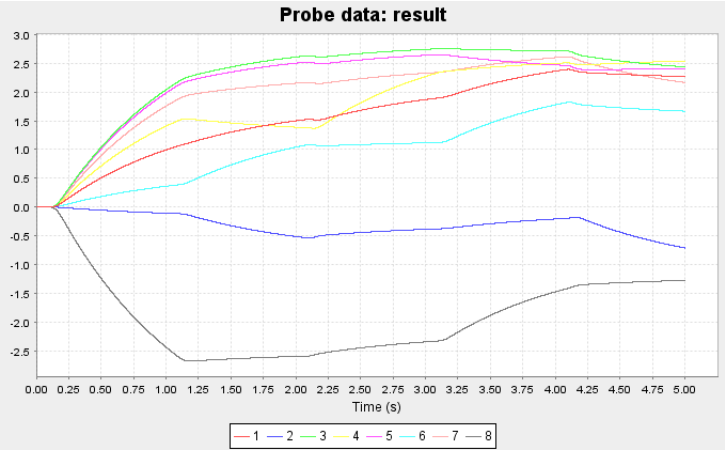


As we can see, the system correctly determines that the correct answer is again number one, three circles. This is despite the fact that we have chosen the possible solutions to be intentionally confusing. The incorrect answers are as close to the correct answer as possible, so the system's ability to differentiate them demonstrates its ability to accurately determine results as opposed to guessing in a general area.

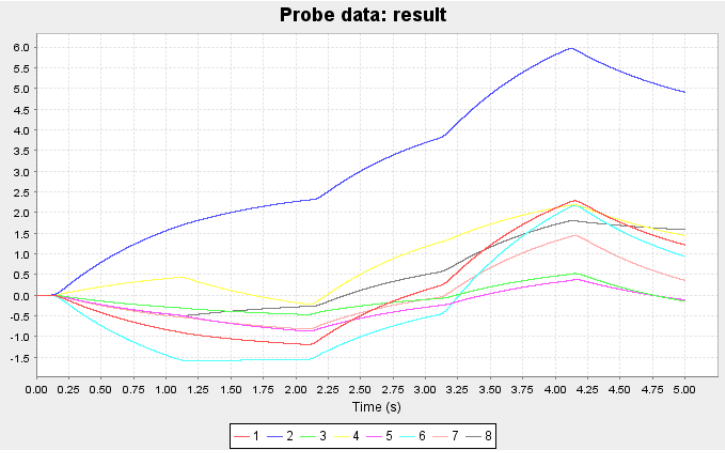
There may be some doubt as to whether the system is actually learning a general rule, or is cheating somehow to find the correct answer. We can test this by changing which signal we present as the second last cell (the function input to the calcLast network). By leaving the input signal the same we ensure that the system will learn an identical transformation, but we are now applying that transformation to different cells to ensure that the transformation actually represents a general rule.



In this test we set the secondLast signal to be one circle. Again, the model found the correct answer of two circles (same shape, plus one) or number 7. It only barely beat out number 6 (four circles), which may not be as confident a result as we would like, but we could increase the accuracy by increasing the dimension of the vectors or presenting the system with more examples to learn from.



In this test we set the secondLast signal to be two squares. In this case the system incorrectly reported that the correct answer was number 4, three diamonds; however, the correct answer of 3 was only slightly behind. We then tried a logically identical test of two triangles:



This time the system correctly decided that the correct answer was number two, three triangles. We suspect that the reason the system was so emphatically correct on this test but wrong on the previous one was the random nature of our vocabulary words. With only 10 dimensions, there is a good chance that the word for "square" could be very similar to the word for "diamond", causing the system to get confused. On the other hand, it looks like in this case the word for triangle was very distinct, making it easy for the system to find the correct answer. We should therefore be able to improve all of our results, and avoid incorrect answers such as in the previous test, by increasing the dimension of the vectors. This is something we will investigate with further testing. Overall, we take these tests as demonstrating that our system can correctly solve simple matrices, and that it does so by finding general rules.

5 Discussion

There are several limitations of our model:

1. It does not realistically simulate the neural properties. Since one of the main goals of our model is neurological plausibility, this is something we intend to address as soon as possible. However, we are confident that this will simply be a matter of time, not requiring any significant changes to our model. One of the advantages of the NEF is that we can easily switch between different levels of detail; all we are doing here is modifying the properties of the neural populations, the overall logic of the system remains unchanged.

2. The matrices our system is solving are very simplified. We would like our model to be able to tackle at least the intermediate problems on Raven's Advanced Progressive Matrices, and that will require expansion of its current abilities. One hypothesis is that there is a "simplifying system" prior to this one in the visual processing stream (this might be what is occurring in the attention directing system in the superior parietal cortex).

This simplifying system would take the overall matrix and break it down into the simple inputs which can be handled by the rule finding system we have developed here.

3. We have not tested our system as thoroughly as we might like. Right now creating test matrices is a somewhat laborious, manual process. We need to streamline the testing system so that we can better examine under what circumstances our system succeeds, and what causes it to fail.

5.1 Future directions

There are a number of enhancements we would like to make to the current model. First, we would like to try encoding T in the synaptic weights rather than representing it explicitly (as seen in Eliasmith [2005]). This should prove useful as we move to higher dimensions, where explicit representation becomes more and more difficult.

Second, we would like to expand our model to cover more of the RPM solving process. Most importantly this would involve modelling the abstract reasoning system (the counterpart to the visual processing system as discussed in the system description). This would allow the system to solve the more difficult matrices that require abstract rules, such as "every shape in a row is different". It would also be interesting to model other visual processes, such as the simplification system mentioned above. This would allow our system to handle more complex inputs, rather than requiring the preprocessing that is currently occurring. We would like to avoid preprocessing as much as possible, because it has a tendency to sneak the solution into the inputs. For example, in Matrix 1 we are encoding the matrix using only shape descriptions; thus the system does not need to determine that what is relevant is shape rather than something like colour, which would otherwise be a tricky problem. Ideally we would like to incorporate a perceptual system into the model so that it can handle the raw matrix image. This is the advantage of building the system in Nengo—if such a perceptual system is developed, it will not be hard to integrate it into our current model.

Third, we would like to expand the system so that it can handle three-cell rules. Under the current model we can only learn pair-wise rules (ex. "the previous cell plus one"), but many RPM problems cannot be solved this way. For example, a common rule is figure addition, where the figures in the first two cells are superimposed in some way to form the third cell. We could not learn this rule under the current system, because the relationship between the first and second cell is not the same as the relationship between the second and third cell. One possible way to get around this problem would be to treat the first and second cells as a single vector, and learn the transformation from that combined vector to the third cell. This may cause problems, because we would only have two training examples in a matrix. We would also have to think carefully about how to structure the representation of those first two cells. It is unlikely that simply adding the representations of the first and second cells

together would give us enough structural information, we would need to clearly distinguish which information came from cell 1 and which from cell 2.

This is related to our final future improvement, which is more complex matrix representations. When representing Matrix 1 and 2 we used a very simple structure of [attribute $\otimes$ value + attribute $\otimes$ value ...]. As we move to more difficult problems involving more complex transformations, this will likely prove insufficient. For example, if we want to add two figures together then we need to know not only what shapes they contain, but where those shapes are. We will need to come up with methods to encode more structure, while balancing the need to keep the size of the vectors (and corresponding neural populations) reasonable.

6 Conclusion

We have presented a mechanistic, neurally plausible model that is consistent with the available behavioural and neurological data on Raven's Progressive Matrices. This model is able to solve simple RPM-like problems, and it does so by finding general rules rather than one-off solutions.

Even with these results, we must keep in mind that this system is nowhere close to being able to sit down and take the RPM. What we have modeled here is one component, the rule-finding system, of the wide network of abilities necessary to take this test.

Despite these limitations, the benefits of this model are several. First, it serves as an initial core, from which we can expand in the future to develop a more comprehensive system. Developing in Nengo facilitates this by employing a standard interface that can be linked with other Nengo models. Second, developing a model forces us to examine our theories, and reveals gaps or inconsistencies therein. In this case we found that we are lacking theories as to how novel rules are generated, which motivated the focus of this model. And finally, the existence of this model demonstrates that neurally based models are a useful avenue to pursue when trying to accomplish these types of high-level reasoning tasks. The fact that we were able to accomplish what we did, even in this limited problem set, with such few and simple components is evidence that neural models are interesting for not just their insight into the brain but also their concrete results. Hopefully this will only become more clear as the model grows and is able to tackle increasingly complex problems.

References

Renée Babcock. Analysis of age differences in types of errors on the Raven's Advanced Progressive Matrices. *Intelligence*, 30:485–503, 2002.

Patricia Carpenter, Marcel Just, and Peter Shell. What one intelligence test measures: A theoretical account of the processing in the Raven's Progressive Matrices test. *Psychological Review*, 97:404–431, 1990.

Richard DeShon, David Chan, and Daniel Weissbein. Verbal overshadowing effects on Raven's Advanced Progressive Matrices: Evidence for multidimensional performance determinants. *Intelligence*, 21:135–155, 1995.

Chris Eliasmith. Cognition with neurons: A large-scale, biologically realistic model of the wason task. In *Proceedings of the 27th Annual Conference of the Cognitive Science Society*, 2005.

Chris Eliasmith and Charles Anderson. *Neural Engineering: Computation, representation, and dynamics in neurobiological systems*. MIT Press, 2003.

Richard Haier, Benjamin Siegel, Keith Nuechterlein, Erin Hazlett, Joseph Wu, Joanne Paek, Heather Browning, and Monte Buchsbaum. Cortical glucose metabolic rate correlates of abstract reasoning and attention studied with Positron Emission Tomography. *Intelligence*, 12:199–217, 1988.

Patrick Kyllonen and Raymond Christal. Reasoning ability is (little more than) working-memory capacity?! *Intelligence*, 14:389–433, 1990.

Brachia Marshalek, David Lohman, and Richard Snow. The complexity continuum in the radex and hierarchical models of intelligence. *Intelligence*, 7: 107–127, 1983.

Maria Meo, Maxwell Roberts, and Francesco Marucci. Element salience as a predictor of item difficulty for Raven's Progressive Matrices. *Intelligence*, 35: 359–368, 2007.

Jane Neumann. *Holistic Processing of Hierarchical Structures in Connectionist Networks*. PhD thesis, University of Edinburgh, 2001.

Tony Plate. *Holographic Reduced Representations*. CLSI Publications, 2003.

Vivek Prabhakaran, Jennifer Smith, John Desmond, Gary Glover, and John Gabrieli. 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, 1997.

Terrence Stewart, Bryan Tripp, and Chris Eliasmith. Python scripting in the nengo simulator. *Frontiers in Neuroinformatics*, 3, 2009.

Nash Unsworth and Randall Engle. Working memory capacity and fluid abilities: Examining the correlation between Operation Span and Raven. *Intelligence*, 33:67–81, 2005.

Tom Verguts and Paul De Boeck. The induction of solution rules in Raven's Progressive Matrices test. *European Journal of Cognitive Psychology*, 14:521–547, 2002.

Tom Verguts, Paul De Boeck, and Eric Maris. Generation speed in Raven's Progressive Matrices test. *Intelligence*, 27:329–345, 2000.