Chapter 4

Biological cognition – syntax

4.1 Structured representations

To this point, we have seen how we can construct semantic pointers: high-dimensional vectors that carry information about the statistical structure of some domain. Constructing these kinds of representations has been accomplished for many years by connectionist neural networks. The main complaint against connectionists, however, has been that these kinds of representations are unhelpful for explaining truly complex – truly cognitive – behavior. It has been forcefully argued that structured representations, like those found in natural language are essential to explaining this domain. As I discuss in more detail in chapter 8, researchers have often moved from that less controversial observation to much stronger claims regarding essential properties of these structured representations (e.g., that they are compositional). However, even if we accept the importance of structured representation for explaining cognition, the ultimate arbiter of whether our theories about such representations is a good one is the behavioral data. The purpose of this chapter is to suggest how semantic pointers can be used in structured representations that are able to explain that data – with no additional assumptions about the essential features of structured representations used for cognition.

A typical structured representation is given by natural language utterances like "The dog chases the boy." The structure underlying this representation is the grammar of English, and the component parts are individual words. In an artificial language, like those typically used in computers, such a phrased may be represented with the structured representation `chases(dog, boy)`. Such a representation is structured because the position in which each term appears determines

its grammatical role. The 'verb' is first, the 'agent' is second and the 'theme' is third. If the order of the elements changes, so does the meaning of the structured representation.

The fact that entire concepts, denoted by words, can be moved into different roles has historically posed a serious challenge for approaches that think of the representation of concepts as being distributed. In addition, because the kind of relation that the concept enters into changes depending on which role it is in, it is not immediately obvious how simple associative relationships can be used to capture such structures. The first of these challenges targets distributed connectionist models, and the second targets both localist and distributed connectionist models. In essence, both challenges boil down to the problem of determining how to take the preferred kind of representation of concepts, and construct a new representation that combines multiple concepts playing different kinds of structural, or syntactic, roles. That is, how can concept representations be bound to roles within a structure? As Baraslou (1999, p. 643) has noticed:

> It is almost universally accepted now that representation schemes lacking conceptual relations, binding, and recursion are inadequate.

Consequently, it has been seen as a major challenge for neurally inspired architectures and theories of cognition to address the issue of how structure can be represented. For instance, Barsalou's own perceptual symbol system theory, while addressing semantics well, has been criticized because the theory (Edelman and Breen, 1999, p. 614)

> leaves the other critical component of any symbol system theory – the compositional ability to bind the constituents together – underspecified.

Barsalou's discussion of 'frames' does not describe how neural representations can be bound into such structures. So, despite recognizing the need for building such structures with binding in neural accounts, no precise computational suggestions have been broadly accepted – and no biologically realistic accounts (i.e. that use noisy spiking neural networks) have been offered.

4.2 Neural binding

4.2.1 Without neurons

Connectionists, being concerned with cognitive modelling, have been working on ways of binding multiple representations since the 1990s. Perhaps the best known early solution is that offered by Paul Smolensky . He suggested that a kind of vector multiplication called a tensor product could perform the necessary binding (Smolensky, 1990). Despite the exotic sounding name, tensor products are a straightforward solution to the problem of trying to multply to vectors, and then later get back one of the vectors given the other (i.e. trying to 'divide' them). It is obvious how to do this if we work with scalars, since we have learned how multiplication works since early childhood: $6 \times 2 = 12$ so, $12 \times \frac{1}{2} = 6$ and $12 \times \frac{1}{6} = 2$. Notice that to compute the unknown element given the other two (i.e., the answer and one of the multiples), we needed to 'invert' the known element and multiply. The question is, what happens if the mathematical objects we have are vectors? What is $(6,3) \times (2,5,4)$?

This is where tensor products come in. They define what we mean by '$\times$' in the context of vectors, so we can have operations that mimic what we are used to with scalars. To distinguish this operation from regular mutliplication, the '$\otimes$' sign is typically used. If the vectors we wanted to multiply were the same length, we might not have to bother with tensor products – we could just multiply the elements. Unfortunately, we can't just multiply if there are more elements in one vector than the other. So, the next obvious solution is to multiply all the elements, which gives the tensor product:

$$\begin{pmatrix} 6 \\ 3 \end{pmatrix} \otimes \begin{pmatrix} 2 & 5 & 4 \end{pmatrix} = \begin{pmatrix} 6\times 2 & 6\times 5 & 6\times 4 \\ 3\times 2 & 3\times 5 & 3\times 4 \end{pmatrix} = \begin{pmatrix} 12 & 30 & 24 \\ 6 & 15 & 12 \end{pmatrix}.$$

This way, if we are given the final matrix and one of the vectors, it is clear how we can recover the other vector.

The relevance to binding was evident to Smolensky. If we represent items with vectors, and structures with collections (sums) of bound vectors, then we can use tensor products to do the binding, and still be able to 'unbind' any of the constituent parts. Consider the previous example of `chases(dog, boy)`. To represent this structure, we need vectors representing each of the concepts and vectors representing each of the roles (I will write vectors in bold): (**dog**, **chase**, **boy**, **verb**, **agent**, **theme**). In the context of the SPA, each of these is a semantic pointer.

To bind one role to one concept, we can use the tensor product:

$$\mathbf{b} = \mathbf{dog} \otimes \mathbf{agent}.$$

Now, the resulting vector **b** can be used to recover either of the original vectors given the other. So, if we want to find out what role the **dog** is playing, we can unbind it by inverting it and mutliplying, just as in the scalar case:

$$\mathbf{b} \otimes \mathbf{dog}^{-1} = \mathbf{agent}.$$

A technical issue regarding what we mean by 'inverting' a vector arises here. Suffice it to say for present purposes that there is a good definition of the necessary inversion called the pseudo-inverse, and I will return to this issue later in a related context (see section ???).

Given a means of binding and unbinding vectors, we also need a means of collecting these bindings together to construct structured representations. Smolensky adopted the standard connectionist approach of simply summing to conjoin vectors. This results in a straightforward method for representing sentences. For example, the structured vector representation of the proposition "The dog chases the boy" is:

$$\mathbf{P} = \mathbf{verb} \otimes \mathbf{chase} + \mathbf{agent} \otimes \mathbf{dog} + \mathbf{theme} \otimes \mathbf{boy}.$$

In the case where all of the role vectors are linearly independent (no one vector is a weighted sum of the others), decoding within this structure is the same as before – just convolve **P** with the inverse of the role. These tensor product representations are quite powerful for representing structure with vectors as elements of the structure. They can be combined recursively to define embedded structures, and they can be used to define standard LISP-like operations that form the basis of many cognitive models, as Smolensky shows.

However, tensor products were never broadly adopted. Perhaps this is because many were swayed by Fodor and McLaughin's contention that this was merely a demonstration that you could implement a standard symbolic system (Fodor and McLaughlin, 1990). As such, it would be unappealing to symbolicists because it just seemed to make models more complicated, and it would be unappealing to connectionists because it seemed to lose the neural roots of the constituent representations. For instance, Baraslou (1999, p. 643) comments that

> Early connectionist formulations implemented classic predicate calculus functions by superimposing vectors for symbolic elements in

> predicate calculus expressions (e.g., Pollack 1990; Smolensky 1990; van Gelder 1990). The psychological validity of these particular approaches, however, has never been compelling, striking many as arbitrary technical attempts to introduce predicate calculus functions into connectionist nets.

I suspect that both of these played a role. But there is also an important technical limitation to tensor products, one which Smolensky himself acklowedges (1990, p. 212):

> An analysis is needed of the consequences of throwing away binding units and other means of controlling the potentially prohibitive growth in their number.

He is referring here to the fact that the result of binding two vectors of lengths n and m respectively, gives a new structure with nm elements. If we have recursive representations, the size of our representations will grow quickly out of hand. That is, they will not *scale* well. This also means that, if we are constructing a network model, we need to have some way of handling objects of different sizes within one network. We may not even know how 'big' a representation will be before we have to process it. Again, it is not clear now to scale a network like this.

I believe it is the scaling issue more than any other that resulted in little adoption of Smolensky's suggestion in the modeling community. However, modelers began to attack the scaling problem head on, and now there are several methods for binding vectors that avoid the issue. For binary representations, there are Pentti Kanerva's Binary Spatter Codes or BSCs (Kanerva, 1994). For continuous representations, there are Tony Plate's Holographic Reduced Representations or HRRs (Plate, 1994). And, also for continuous representations, there are Ross Gayler's Multiply-Add-Permute or MAP representations (Gayler, 1998). In fact, all of these are extremely similar, with BSCs being equivalent to a binary version of HRRs and both HRRs and BSCs being a different means of compressing Smolensky's tensor product (see figure 4.1).

Noticing these similarities, Gayler suggested that the term "Vector Symbolic Architectures" be used to describe this class of closely related approaches to encoding structure using distributed representations (Gayler, 2003). Like Smolensky's tensor product representations, each VSA identifies a binding operation and a conjoining operation. Unlike tensor products, the newer VSAs do not change the dimensionality of representations when encoding structure.

$$\textcircled{1} \quad \begin{pmatrix} A \\ B \\ C \\ D \end{pmatrix} \otimes \begin{pmatrix} E \\ F \\ G \\ H \end{pmatrix} = \begin{pmatrix} AE & AF & AG & AH \\ BE & BF & BG & BH \\ CE & CF & CG & CH \\ DE & DF & DG & DH \end{pmatrix}$$

$$\textcircled{2} \quad \begin{pmatrix} A \\ B \\ C \\ D \end{pmatrix} \times \begin{pmatrix} E \\ F \\ G \\ H \end{pmatrix} = \begin{pmatrix} AE \\ BF \\ CG \\ DH \end{pmatrix}$$

$$\textcircled{3} \quad \begin{pmatrix} 1 \\ 0 \\ 1 \\ 0 \end{pmatrix} \oplus \begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \\ 1 \\ 0 \end{pmatrix}$$

$$\textcircled{4} \quad \begin{pmatrix} A \\ B \\ C \\ D \end{pmatrix} \circledast \begin{pmatrix} E \\ F \\ G \\ H \end{pmatrix} = \begin{pmatrix} AE + BH + CG + DF \\ AF + BE + CH + DG \\ AG + BF + CE + DH \\ AH + BG + CF + DE \end{pmatrix}$$

Figure 4.1: Various vector binding operations. (1) Tensor products are computed as a multiplication of each element of the first vector with each element of the second vector. (2) Piecewise multiplication, used in Gayler's Multiply-Add-Permute representation, multiplies each element of the first vector with one corresponding element in the second vector. (3) Binary Spatter Codes (BSCs) combine binary vectors using an exclusive or (XOR) function. (4) Circular convolution is the binding function used in Holographic Reduced Representations (HRRs). An overview of circular convolution and a derivation of its application to binding and unbinding can be found in section B.1.

There is a price for not changing dimensionality during binding operations: slow degredation of the structured representations. If we constantly encode more and more structure into a vector space that does not change in size, we must eventually lose information about the original structure. Representations that slowly lose information in this manner have been called 'reduced' representations, to indicate that there is less information in the resulting representation than in the original structures they encode (Hinton, 1990). Thus the resulting structured representations does not explicitly include all of the bound components: if they did, there would be no information reduction. This information reduction has important consequences for any architecture, like the SPA, that employs them.

For one, this means that such an architecture cannot be a classical architecture. In short, the reason is that the representations in such an architecture are not perfectly compositional. This is a point I will return to in chapter 10. A more technical point is that, as a result of losing information during binding, the unbinding operation returns an *approximation* of the originally bound elements. This means that the results of unbinding must be 'cleaned-up' to a vector that is familiar to the system. I return to this important issue in section B.3. But, what is clear even without detailed consideration of clean-up, is that there are limits on the amount of embedded structure that a reduced representation can support before noise makes the structure difficult to decode, resulting in errors. As we will see, the depth of structure that can be encoded depends on the dimensionality of the vectors being used, as well as the total number of symbols that can be distinguished. Ultimately, however, I think it is this capacity for error that makes the SPA psychologically plausible. This is because the architecture does not define an idealization of cognitive behavior like classical architectures tend to, but rather specifies a functional, implementable system that is guaranteed to fail. I take it that capturing the failures as well as the successes of cognitive systems is truly the goal of a cognitive theory. After all, many experiments in cognitive psychology are set up to increase the cognitive load to determine what kinds of failure arise.

Now to specifics. In this book I adopt Plate's HRR representations because they work in continuous spaces, as neurons do, and because he has established a number of useful theoretical results regarding their capacity limitations (Plate, 2003). Still, most of the examples I provide do not depend on the VSA chosen. In fact much work remains to be done on the implications of this choice. In particular, it would be interesting to determine in more detail the expected behavioral differences between HRRs and MAPs. But, such considerations are beyond the scope of the present discussion. It is worth noting, however, that the structure of the SPA itself does not depend on the particular choice of a VSA. I take any

method for binding that depends on a piecewise (i.e., local) nonlinearity in a vector space to be able to support the main ideas behind the SPA. Nevertheless, it is necessary to choose an appropriate VSA in order to provide specific examples.

In the remainder of this chapter, I turn to the consideration of implementing the architectural components necessary to support HRR processing in a biologically realistic setting. First, I consider binding and unbinding. I then turn to demonstrating how these operations can be used to manipulate (as well as encode) structured representations, and I show how a spiking neural network can learn such manipulations. I then return to the issue of implementing a clean-up memory in neurons, and discuss some capacity results that suggest the SPA will scale appropriately. Finally, I describe a detailed SPA model that performs context-sensitive linguistic inference, mimicking human performance in the classic Wason card selection task.

- relate this to the 'vector processing' section earlier

4.2.2 With neurons

The relevance of VSAs to what the brain does is far from obvious. As Bob Hadley (2009) has recently noted: "It is unknown whether or how VSAs could be physically realized in the human brain" (p. 527). While it might seem obvious at this point how VSAs can be implemented in the brain, this is largely because I have been directing the discussion for exactly this purpose (yes, you have been set up).

To perform binding with neurons, we can simply combine the above characterization of vector binding with my earlier characterizations of vector representation (section 2.5) and transformation (section 3.7). Notably, each of the VSA methods for binding rely on an element-wise nonlinearity (see figure 4.1), so we need to perform a nonlinear transformation (the same one we did in the last tutorial; section 3.7). In the case of HRRs in particular, we need to perform a linear transformation before the element-wise nonlinearity.

Specifically, the binding operation for HRRs is called 'circular convolution.' The details of circular convolution can be found in appendix B.1. What is important for our purposes is that the binding of any two vectors $\mathbf{A}$ and $\mathbf{B}$ can be computed by

$$\mathbf{C} = \mathbf{A} \otimes \mathbf{B} = \mathbf{F}^{-1}(\mathbf{FA}.\mathbf{FB})$$

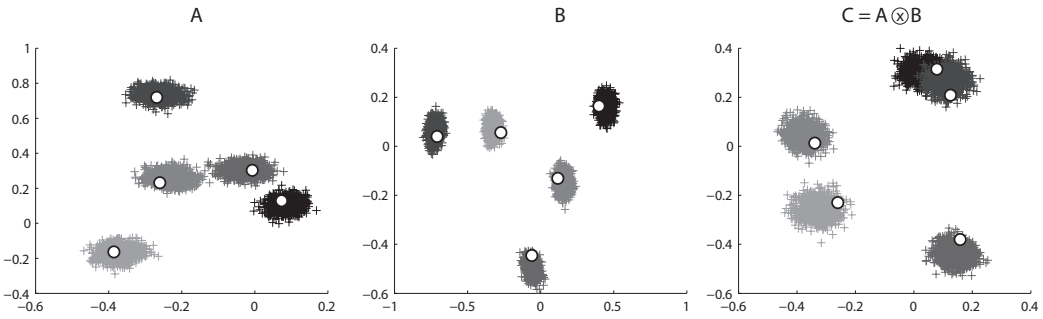


Figure 4.2: Binding vectors with neurons. Two vectors, **A** and **B**, are bound together to form vector **C**. The vectors are all ten-dimensional and are visualized here by graphing the first five dimensions of a vector against its latter five dimensions. Each blob in the graph thus represents two dimensions of the given vector. The spread of the points is due to the variability in the value encoded by the neural representation over a period of one second. The white circles with black borders indicate the ideal value of each vector.

where '.' is used to indicate element-wise multiplication of the two vectors. The matrix **F** is a linear transformation that is the same for all vectors of a given dimension. The resulting vector **C** has the same number of dimensions as **A** and **B**.

Because we have already seen examples of both linear transformation and the multiplication of two numbers, it is straightforward to build this network in Nengo. The tutorial at the end of this chapter gives detailed instructions on building a binding network to encode structured representations (section 4.8). Figure 4.2 shows the results of binding various six-dimensional vectors using the circular convolution method.

To unbind vectors, the same operation can be used. However, as explained earlier we need to compute the inverse of one of the bound vectors to get the other. Conveninently, for HRRs a good approximation to the inverse can be found by a simple linear transformation (see appendix B.1). Calling this transformation **S**, we can write the unbinding of any two vectors as

$$\mathbf{A} \approx \mathbf{C} \otimes \mathbf{B}^{-1} = \mathbf{F}^{-1}(\mathbf{FC}.\mathbf{FSB}).$$

Recall that the result is only approximate and so must be cleaned-up, as I discuss in more detail shortly (section B.3).

From a network construction point-of-view, we can redeploy exactly the same network as in the binding case, and simply pass the second vector through the

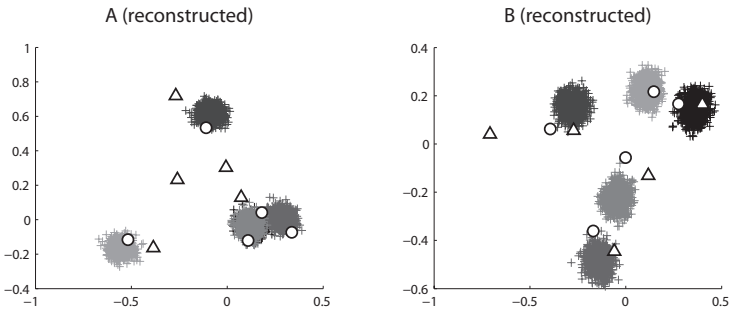


Figure 4.3: Unbinding vectors with neurons. The reconstructed vectors **A** and **B** were obtained by using the transformations $\mathbf{A} \approx \mathbf{C} \otimes \mathbf{B}^{-1}$ and $\mathbf{B} \approx \mathbf{C} \otimes \mathbf{A}^{-1}$ respectively. The original ideal vectors are the same as those used in figure 4.2 and are shown here as white triangles with black borders. The circles in this figure outline the analytically calculated unbinding; that is, the result of using the unbinding equations directly, without a neural implementation. Both the neural representation and the analytical result are close to the vectors originally bound, but there is a significant amount of noise introduced in the unbinding process that requires clean-up.

transformation **S** before presenting it to the network. Figure 4.3 demonstrates, using the output of the network in figure 4.2, that vectors can be effectively unbound with just the addition of this linear transformation.

It is perhaps unsurprising that we can implement binding and unbinding in neural networks. However, this does not allay concerns, like those expressed by Barsalou, that this is merely a technical trick of some kind that has little psychological plausibility. I believe the psychological plausibility of these representations need to be addressed by looking at larger scale models, the first of which we will see later in this chapter (section 6.5.2). However, a similar concern about the neural plausibility of this approach can be addressed by looking at these simpler networks.

The three possible sources of concern regarding neural plausibility are the nonlinearity, the connectivity, and how well the networks scale. Let me consider each in turn. The nonlinearity that is needed for the binding operation is computing the product of two scalar input signals. We have demonstrated in the tutorials how this can be done accurately using only linear dendrites, and with a small number of neurons (section 3.7). The assumption of linear dendrites is, in fact, a limiting assumption, not a beneficial one. We make it here because it

is the 'textbook' assumption about dendritic processing. However, much recent work strongly suggests that pyramidal cells (the most common cortical cells) have nonlinear dendritic interactions (???refs from albert).

- paragraph from Albert: nonlinearities everywhere, use Albert's research on this; would be super easy to implement that way but can do with linear operations too... this discussion is referred to by last tutorial, and would be useful for the next chp on control as well!

If our binding networks employed such nonlinearities, we could eliminate the middle layer currently used to compute the nonlinearity (see figure ???). Consequently, the binding networks would use significantly fewer neurons than in the examples we consider here.[1] So, the examples we present are worst case scenarios for the number of neurons that are needed to generate structured representations.

The second concern regarding neural plausibility is connectivity. That is, we know that in general cortical circuits are fairly locally and sparsely connected (Song et al., 2005; Hellwig, 2000; Lund et al., 1993). Do these binding networks respect this constraint? To begin, we need to be more accurate about what we mean by locally connected. In their anatomical work that involves microinjections of a tracer across several areas of monkey cortex, the Levitt group has found that small injections of 100μm (i.e., 0.1mm) spread to areas of approximately 3mm by 3mm or slightly larger (Lund et al., 1993). ???There is an interesting structure to this projection pattern, in which there are patches of about 300-400μm in diameter separated by spaces of a similar size. These patches correspond well to the sizes of dendritic fields of the pyramidal cells in these areas.???

- Todo: see if the pattern between ??? can be matched by a binding network, or how that fits. In the paper, they suggest that the spacing is partly because of inhibitory neurons that are surrounding the pyramidal cells (basket cells, which have arbors of about 1mm in diameter). need to use dale's principle stuff to test this out too... only about 40K neurons in the little patches.

This pattern of connectivity has been found throughout visual, motor, somosensory and prefrontal cortex, and across monkeys, cats, and rodents.

The plausibility of connectivity is closely tied to the third issue of scaling. To determine if a binding network can be plausibly fit within this cortical area,

[1] We have constructed the same network with both linear and nonlinear neurons to compare the differences. The nonlinear networks, as expected, are much more resource efficent and more accurate (ref bruce???).

it is important to know what the dimensionality of the bound vectors is going to be. It is also important to know how the size of the binding network changes with dimensionality. I discuss in detail the dimensionality of the bound vectors in section B.3, because the necessary dimensionality is determined by how well we can clean up vectors after binding. There I show that we need vectors of about 2,400 dimensions in order to capture large structures with an adult-sized vocabulary. It is also clear that the binding network itself will scale linearly with the number of dimensions because the only nonlinearity needed is the element-wise product. Thus, one additional dimension means computing one additional product. Recall from section 3.7 that 100 neurons per dimension results in good quality estimates of the product. Taken together, these considerations suggest that about 240,000 neurons are needed to bind two vectors.

There are about 170,000 neurons per mm^2 of cortex,[2] so this suggests that we need just under 1.5 mm^2 of cortex to perform the necessary binding. The projection data suggests that local projections cover at least 9 mm^2 of cortex, so the binding layer can fit comfortably within this area. If we want to include the two input populations, the binding layer and the output population with similar assumptions, we would need about 6 mm^2 of cortex. Recall that the unbinding network requires the same resources. Consequently, these networks are consistent with the kind of local connectivity observed in cortex. As well, the architecture scales linearly so this conclusion is not highly sensitive to the assumptions made regarding the dimensionality of the representations. It is worth noting that if we allow dendritic nonlinearities, the binding layer is not needed, so the entire network would require only 4.5 mm^2 of cortex.

Together, these considerations suggest that the vector binding underlying the SPA can be performed on the scale necessary to support human cognition in a neurally plausible architecture. I return to considerations of the neural plausibility of the SPA in chapter 10.

4.3 Manipulating structured representations

To this point, my discussion on syntax in the SPA has focussed on how structures can be encoded and decoded using binding. However, to make such representations cognitively useful they must be *manipulated* to support reasoning.

[2] This estimate is based on the data in Pakkenberg and Gundersen (1997) which describes differences in density across cortical areas, as well as the effects of age and gender. This value was computed from Table 2 for females, using a weighted average across cortical areas.

An interesting feature of VSA representations is that they can often be manipulated without explicitly decoding the elements. This has sometimes been called 'holistic' transformation, since a semantic pointer representing a sentence can be manipulated in a way that affects the entire sentence at once without manipulating the entities it encodes.

Several researchers have demonstrated transformations of structured representations using distributed representations, although experiments have been limited to fairly simple transformations and the transformations tend to be hand-coded.[3] Here, I begin by considering simple transformations as well, but demonstrate in the next section how transformations can be learned. As well, throughout the remainder of the book subsequent examples will use progressively more sophisticated transformations.

Conveniently, transformations using VSAs rely on binding operations as well. Conseqeuently, no additional network elements are needed to support syntactic manipulation. To see how these manipulations work, let us return to the `chases(dog, boy)` example, where this sentence is encoded as

$$\mathbf{P} = \mathbf{verb} \otimes \mathbf{chase} + \mathbf{agent} \otimes \mathbf{dog} + \mathbf{theme} \otimes \mathbf{boy}.$$

Recall that given this encoding, we can decode elements by binding it with the appropriate inverses. For instance, if we multiply **P** by the approximate inverse of **chase** (i.e., **chase'**) we get:

$$\begin{aligned}
\mathbf{chase}' \otimes \mathbf{P} &= \mathbf{chase}' \otimes (\mathbf{verb} \otimes \mathbf{chase} + \mathbf{agent} \otimes \mathbf{dog} + \mathbf{theme} \otimes \mathbf{boy}) \\
&= \mathbf{chase}' \otimes \mathbf{verb} \otimes \mathbf{chase} + \mathbf{chase}' \otimes \mathbf{agent} \otimes \mathbf{dog} + \mathbf{chase}' \otimes \mathbf{theme} \otimes \mathbf{boy}) \\
&= \mathbf{verb} + \mathbf{noise} + \mathbf{noise} \\
&\approx \mathbf{verb}
\end{aligned}$$

A few comments are in order. First, this is an algebraic demonstration of the effects of binding **chase'** to **P**, and does not require the decoding of any of the elements. Second, this demonstration takes advantage of the fact that $\otimes$ can be treated like regular multiplication as far as allowable operations are concerned (i.e., it is communitative, distributive and associative). Third, I have used the $'$

[3] Niklasson and van Gelder (1994) demonstrated how to transform a logical implication into a disjunction, as did Plate (1994); Pollack (1990) transformed reduced representations of the form *loved*(*x*,*y*) to the form *loved*(*y*,*x*); Legendre et al. (1994) showed how to transform representations for active sentences into representations for passive sentences (and vice-versa).

sign to indicate the pseudo-inverse and distinguish it from a true inverse, which is computed by a shift as mentioned earlier (section 4.2.2). Fourth, $\mathbf{chase}' \otimes \mathbf{verb} \otimes \mathbf{chase}$ is replaced by **verb** because a vector times its inverse is equal to one. Or, more accurately, a vector times its pseudo-inverse is approximately equal to one, and any difference is absorbed by the **noise** term. Finally, the last two terms are treated as noise because circular convolution has the important property that it maps the elements to an approximately orthogonal result. That is, the result $\mathbf{w} = \mathbf{x} \otimes \mathbf{y}$ is unlike either **x** or **y**. By 'unlike' we mean that the dot product (a standard measure of similarity) between **w** and **x** or **y** is close to zero. Consequently, when we add in these new but unfamiliar items from the last two terms, they will slightly blur the correct answer, but not make it unrecognizable. In fact, the reason we really think of these as noise is because they will not be similar to any known lexical item (because they are orthogonal to known items), and hence during the clean-up operation will be ignored. Once again, it is clear that clean-up, while not an explicit part of most VSAs, plays a central role in determining what can be effectively represented and manipulated. This is why it is a central part of the SPA.

We could think of this simple decoding as a kind of manipulation of the structure, but we can also perform more extensive manipulations, such as switching roles and changing the relation, in a similar manner. Consider the hand-constructed vector **T**:

$$\mathbf{T} = \mathbf{agent}' \otimes \mathbf{theme} + \mathbf{chase}' \otimes \mathbf{hug} + \mathbf{theme}' \otimes \mathbf{agent}$$

Convolving our original representation of the sentence **P** with **T** produces:

$$\mathbf{T} \otimes \mathbf{P} = \mathbf{verb} \otimes \mathbf{hug} + \mathbf{agent} \otimes \mathbf{boy} + \mathbf{theme} \otimes \mathbf{dog} + \mathbf{noise}.$$

That is, we now have a representation of `hug(boy, dog)`. This occured because the first term in **T** converts whatever was the **agent** into the **theme** by removing the **agent** vector (since $\mathbf{agent}' \otimes \mathbf{agent} \approx 1$) and binding the result with **theme**. The last term works in a similar manner. The second term replaces **chase** with **hug**, again in a similar manner. The **noise** term is a collection of the noise generated by each of the terms.

For a demonstration of this ability, we can consider one of the standard tasks for testing cognitive models: question answering. The task of question answering is often a straightforward application of the transformation function. Figures 4.4 and 4.5 show a network of 10,000 neurons answering four different questions about different sentences. For each sentence and question, a different input is

provided to the same network for 0.25 seconds. In figure 4.5, the results of clean up are shown, so the similarity between the output of the network and possible responses is plotted. The system is considered to have given a correct response if its output is more similar to the correct response than any other possible responses.

The first case is nearly identical to the first transformation we considered above. Instead of determining the role of **chase**, the network determines what the action of the sentence is. For more complex structures, we need to suggest ways in which a linguistic sentence are mapped into a semantic pointer. These decisions are important, but it is not immediately obvious what the best method for doing so is. For present purposes, to demonstrate more sophisticated question answering consider "The dog chasing the boy caused the boy to fall". In such a sentence the roles such as 'agent' and 'theme' will be multiply instantiated. Consequently, roles need to be tagged with the verb they are roles for. This can be accomplished through binding as follows:

$$\begin{aligned}\mathbf{P}_2 \;=\; & \mathbf{verb}\otimes\mathbf{cause}+\mathbf{agent}\otimes\mathbf{cause}\otimes(\mathbf{verb}\otimes\mathbf{chase}+\mathbf{agent}\otimes\mathbf{chase}\otimes\mathbf{dog}+\\ & \mathbf{theme}\otimes\mathbf{chase}\otimes\mathbf{boy})+\mathbf{theme}\otimes\mathbf{cause}\otimes(\mathbf{verb}\otimes\mathbf{fall}+\mathbf{agent}\otimes\mathbf{fall}\otimes\mathbf{boy})\end{aligned}$$

This additional 'tagging' of elements by their verb does not affect previous results and was not discussed for the sake of simplifying the exposition. Here, as in many more sophisticated linguistic structures, it is essential to distinguish different roles in order to preserve the appropriate structure. If the roles were not tagged, it would be more difficult to distinguish whether **dog** was the agent of **chase** or of **fall**.

The second example in 4.4 asks the question "What caused the boy to fall?" by convolving $\mathbf{P}_2$ with $\mathbf{agent}\otimes\mathbf{cause}'$ and the correct answer is provided. A more challenging kind of question queries the contents of subphrases that make up the sentence, e.g. "Who fell?". This question is asked in example three by determining the $\mathbf{agent}\otimes\mathbf{fall}$ of the $\mathbf{theme}\otimes\mathbf{cause}$ all at once. That is,

$$\begin{aligned}&\mathbf{P}_2\otimes(\mathbf{theme}\otimes\mathbf{cause}\otimes\mathbf{agent}\otimes\mathbf{fall})'\\ \approx&\mathbf{boy}\end{aligned}$$

The same answer would result from applying the two transformations in series. It should be noted, however, that again challenges arise when considering to how to map questions onto the appropriate transformations. I will return to this issue shortly.

First, we can consider the fourth example from figure 4.4. Here the system answers the question "What is beside the big star?" for the sentence "The big star

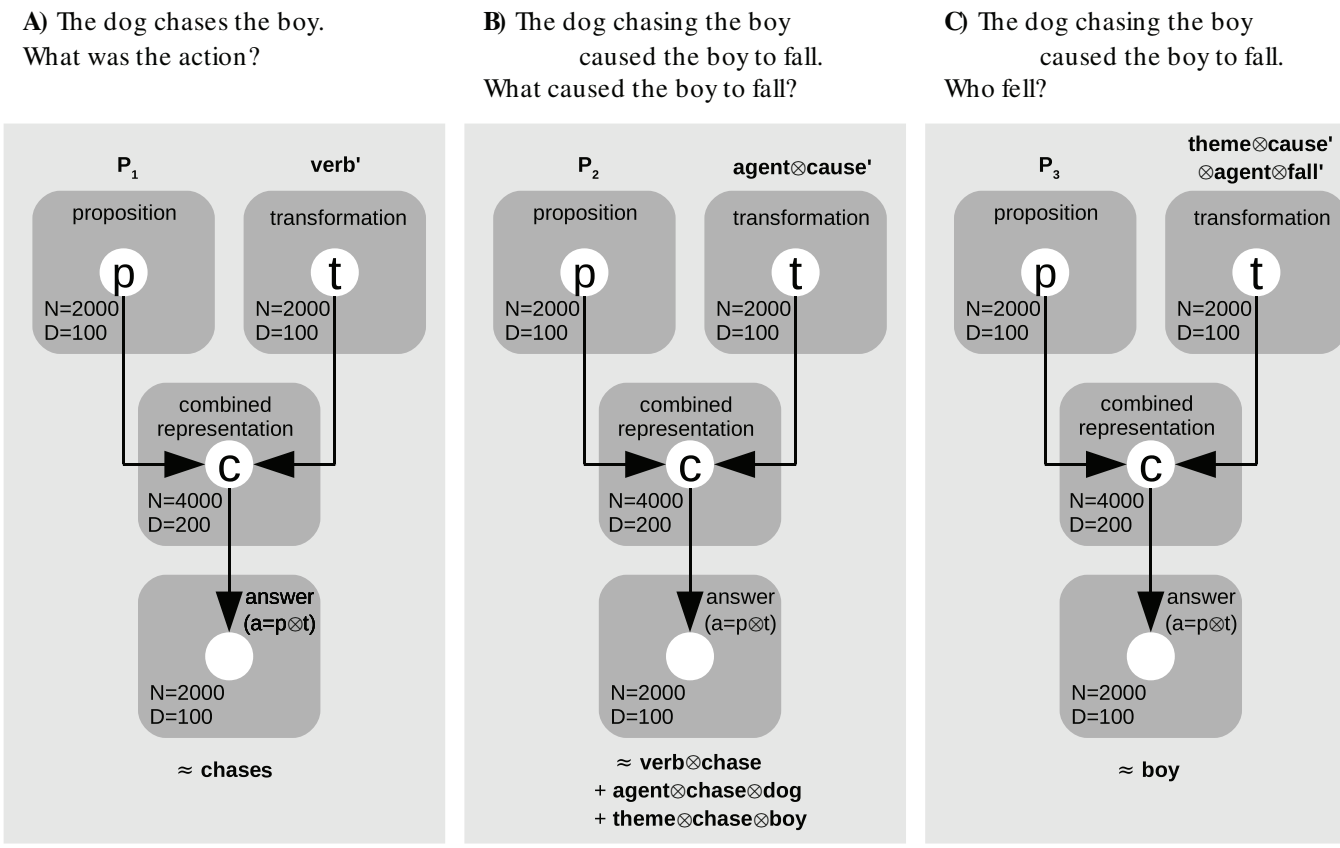


Figure 4.4: A question-answering network handling four different questions. For each case, the sentence and question are provided as separate inputs (p and t). Since the synaptic connections do not need to be modified, this one network is capable of answering any question of this form.

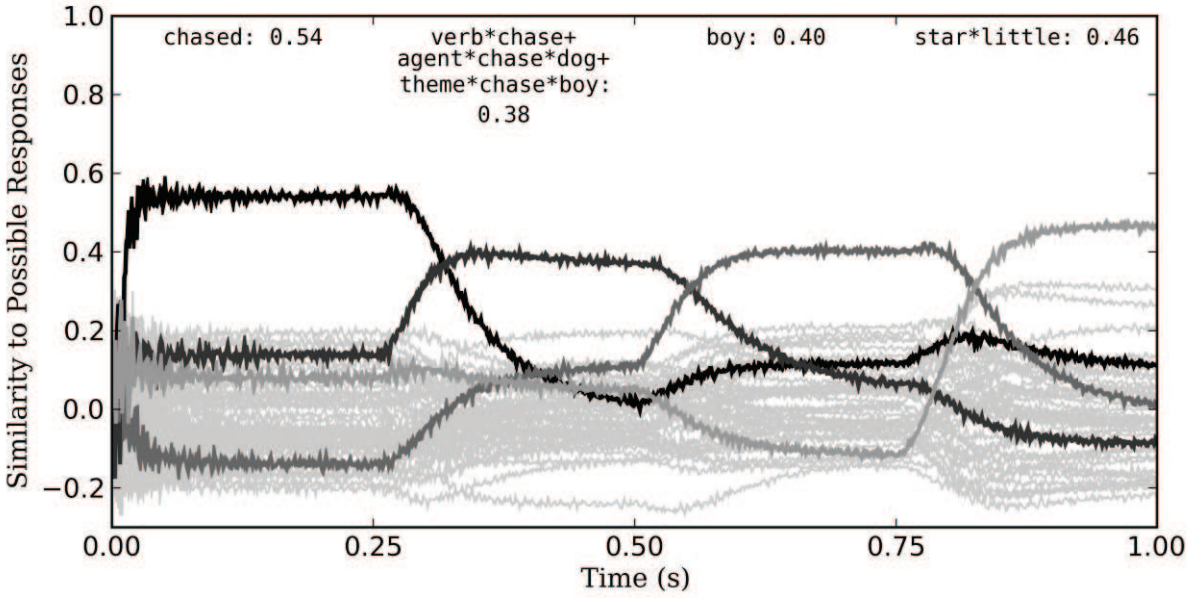
is beside the little star." This example was chosen to demonstrate that there is no inherent 'problem of two' for this approach (see section 1.3). This is not surprising since representation of the same object twice does not result in representations that cannot be distinguished. This example is structurally identical to the first example, with the difference that there are modifiers ('big' and 'little') bound to the nouns.

To conclude this section, it is worth noting two important challenges that are raised by considering the manipulation of language-like representations. First, there is the issue of mapping language onto a VSA representation like that used by the SPA. There are important decisions regarding how we should represent complex structure that are insufficiently addressed here. The second closely related challenge is how natural language questions are mapped to transformations of those structures. There are multiple possible ways to affect such transformations. In this chapter I have been considering transformations that are bound to the representations in one step. However, it is quite likely that in many circumstances, the way questions are answered is partly determined by how representations are moved through the system. If the question demands deep semantic analysis, for instance, then the employed pointers need to be dereferenced. A simple syntactic manipulation will not suffice. Consequently, the next chapter, on control structures in the SPA, is relevant to how such question-answering is performed. In short, I want to be clear that the treatment provided here is self-consciously superficial. I do not want to suggest that all reasoning can be accomplished by syntactic manipulation, even though such manipulation is important to much of language-based cognition. I return to both of these issues in chapter 10.4.

4.4 Learning structural manipulations

Unlike most connectionist approaches, the NEF does not rely on a learning process to construct model networks.[4] This is demonstrated, for example, by the binding networks derived above. However, learning clearly plays a large role in cognition, and is important for explaining central features of cognition such as

[4]There is much to recommend not having learning be a sole network design method. It makes it possible to avoid some of the unprincipled aspects of learning, such as how to distinguish example and test elements, how to order the examples, when to turn learning on and off, what learning rates to choose, etc. It is also easier to test specific ideas about what a brain area is doing, to design large-scale and many layered networks, and to take advantage of established empirical facts about the area under study. Of course, if you want to answer certain developmental questions, or questions related to certain kinds of adaptation of the system, careful consideration of learning is unavoidable. I address learning in chapter 6.



place

holder

Figure 4.5: The network answering four questions. Each question is presented for 0.25 seconds by setting the input currents to the neural groups p and t from figure 4.4. The resulting spiking behavior of neural group a is interpreted as a semantic pointer and compared to the four correct responses (dark gray through light gray). Also shown (lightest gray) are the similarities of the output with 40 randomly chosen other semantic pointers, representing other possible responses. Since the output is closer to the correct response than any other answer, this network successfully answers each question.

syntactic generalization and fluid intelligence (see sections 4.6 and 6.5.2). So let us consider how to learn structural manipulations.

In my previous example of structural manipulation, as in most past work on syntactic transformations, the transformation vectors **T** are hand-picked based on previous knowledge of the vectors, and the ways in which they are combined. This is generally undesirable from a cognitive modelling perspective, since it requires significant design decisions on the part of the modeller, and these tend to be driven by particular examples. It's even more undesirable from a psychological perspective because one of the distinctive features of cognition is the ability to generate new rules given past experience. Most commonly, this is called *induction*.

Inductive reasoning is the kind of reasoning that proceeds from several specific examples to a general rule. Unlike deductive reasoning, which forms the foundation of classical logic, and guarantees the truth of any correctly derived conclusion, inductive reasoning can be correctly used to generate rules that may well be false. Nevertheless, induction is perhaps the most common kind of reasoning used both in science, and in our everyday interactions with the world.

In the language of the SPA, performing induction is equivalent to deriving a transformation that accounts for the mapping between several example structure transformations. For example, if I tell you that "the dog chases the boy" maps to "the boy chases the dog", and that "John loves Mary" maps to "Mary loves John" and then provide you with the following structure: "the bird eats the worm"; you would probably tell me that the expected mapping would give "the worm eats the bird" (semantic problems notwithstanding). In solving this problem, you used induction over the syntactic structure of the first two examples to determine a general structural manipulation that you then apply to a new input. We would like our architecture to do the same.

Neumann (2001) presented an early investigation into the possibility of learning structural transformations using a VSA. She demonstrated that with a set of example transformations available, a simple error minimization would allow extraction of a transformation vector. In short, she showed that if you had examples of the transformation, you could infer what the transformation vector is by incrementally eliminating the difference between the examples, and the transformations you calculated with your estimate of **T** (see appendix B.2).

A simple extension to her original rule that allows the transformation to be calculated on-line looks like this:

$$\mathbf{T}_{i+1} = \mathbf{T}_i - w_i \left[\mathbf{T}_i - \left(\mathbf{A}_i' \otimes \mathbf{B}_i \right) \right].$$

In this equation, i indexes the example, **T** is the transformation vector we are

trying to determine, w is a weight that determines how important we take the current example to be, and **A** and **B** are the pre- and post-mapping structured vectors . This rule essentially estimates the transformation vector from **A** to **B** (by performing the convolution between the inverse of the input and the result; i.e. assuming $\mathbf{B} = \mathbf{T} \otimes \mathbf{A}$), subtracts that from the current estimate of the overall transformation vector $\mathbf{T}_i$, and then uses that difference to update the current estimate of the overall transformation.

This simple rule is surprisingly powerful, as I show in the example in section 4.6. I demonstrate the rule in the context of a model of fluid intelligence, and in order to describe all of the components of that model, I need to return to the issue of clean-up memory.

4.5 Clean-up memory and scaling

Once a syntactic transformation has been learned, it can be applied to a novel structured representation. However, the result – as emphasized in the earlier discussion – will be noisy. This is true for any compressed representation, like those employed in VSAs. In the case of HRRs, the noisy results are in a high-dimensional continuous vector space, and they must be 'cleaned up' to the nearest valid representation. Most typically, we can think of 'valid' representations as those vectors which are associated with words in a lexicon. However, they might also be sentences, subphrases, or other structured representations themselves. Ultimately, what makes a representation valid is its inclusion in a clean-up memory.

As is clear from the examples in section 4.3, the more complex the manipulation, and the more elements in a structure, the more noise there will be in the results of a manipulation. Consequently, to demonstrate that the SPA is feasible in its details, it is crucial to show that a scalable, effective clean-up memory can be constructed out of biologically plausible neurons. Terry Stewart and Charlie Tang have worked out the details of such a memory in my lab (Stewart et al., 2010). Here I describe some of the highlights of that work. I begin by describing the clean-up memory itself and then turn to issues of scaling.

Mapping a noisy or partial vector representation to an idealization is a function that plays a central role in many neurally inspired cognitive models (Sun, 2006; Anderson and Lebiere, 1998; Pollack, 1988). As well, there have been several suggestions as to how such mappings can be done, including Hopfield networks, multilayer perceptrons, or any other prototype-based classifier: in short, any type of auto-associator.

In the SPA, an ideal clean-up memory would take a given semantic pointer **A** and be able to correctly map any vector extracted from it to a valid lexical item. For present purposes, we can suppose that **A** is the sum of some number of slot-filler pairs. Algorithmically, decoding this would be described as:

1. Convolve input **A** with a probe vector **p** (done with a convolution network)
2. Measure the similarity of the result with all valid items
3. Pick the item with the highest similarity and return that as the cleaned up result

The difficult computation in this algorithm is measuring the similarity with all valid items. This is because the many-to-one mapping between noisy input and clean output can be complicated.

So, unfortunately, many simple auto-associators, including linear associators and multilayer perceptrons, do not perform well (Stewart et al., 2010). These are simple because they are feedforward: the noisy vector is on the input, and after one pass through the network, the cleaned up version is on the output. Better associators are often more complicated. The Hopfield network, for instance, is a dynamical system that constantly feeds its output back to the input. Over time, the system settles to an output vector that can be considered the cleaned up version of an input vector. Unfortunately, such recurrent networks often require several iterations before the results are available, slowing down the over all system. Ideally, we would like a clean-up memory that is both fast, and accurate.

To build such a memory, we can exploit two of the intrinsic properties of neurons described in section 2.1, and demonstrated in the tutorial on neural representation (section 2.5). The first is that the current in a neuron is the dot-product of an input vector with the neuron's preferred direction in the vector space. The dot product is a standard way to measure the similarity of two vectors. So, a similarity measure is a natural neural computation. The second property is that neurons have a non-linear response, so they do not respond to currents below some threshold. This means they can be used to compute nonlinear functions of their input. Combining a similarity measure and a nonlinear computation turns out to be a good way to build a clean-up memory.

Specifically, for each item in the clean-up memory, we set a small number of neurons to have a preferred direction vector that is the same as that item. Then if that item is presented to the memory, those neurons will be highly active. And, for inputs near that item, these neurons will also be somewhat active. How near

items must be to activate the neurons will depend on the firing thresholds of the neurons. Setting these to be slightly positive in the direction of the preferred direciton vector will make the neuron insensitive to inputs that are only slightly similar. In effect, the inherent properties of the neurons are being used to clean up the input.

Figure 4.6 shows how this neural clean-up memory scales. The parameters affecting how good the clean-up is are: k, the number of pairs of bound items in the input vector; M the number of valid lexical items; and D the dimensionality of the vector. In the simulations shown here, performance for values up to $k = 8$ are shown because George Miller's (1956) classic standard limits of seven plus or minus two on working memory fall in about the same range. The graphs show M up to a value of 60,000 because this is approximately the size of an adult lexicon (Crystal, 2003). And, we can see the values of D that allow for 99% accuracy over these ranges of k and M.

- As Geoff Hinton recently wrote: "In the Hitchhiker's Guide to the Galaxy, a fearsome intergalactic battle fleet is accidentally eaten by a small dog due to a terrible miscalculation of scale. I think that a similar fate awaits most of the models proposed by Cognitive Scientists" p. 7 (red handout).

While it would be difficult to argue that we are in a position to definitively set a 'maximum' number of dimensions for semantic pointers in the SPA, these simulations provide good evidence that the architecture is scalable to within the right neighborhood. Because this model is constructed with 10 neurons per lexical item, we need approximately 600,000 neurons to implement the best clean-up memory model described here. This works out to about 3.5mm^2 of cortex, which fits well within the connectivity patterns discussed in section 4.2.2. In addition, the connectivity matrix needed to implement this memory is of the same dimensionality as in the binding network, and so respects the anatomical constraints discussed there as well. So, when coupled with the previous discussion regarding the neural plausiblity of the binding network (section 4.2.2), there is a strong case to be made that with about ??? dimensions, structure representation and processing of human-like complexity can be accomplished by an SPA in the brain. This is not to say we have a model that performs all human-like structure processing, but rather to say that we have good reason to suppose we have appropriate tools for constructing such a model. We will see an example of such a model in the next section.

First, however, it is worth considering a few issues related to this clean-up memory discussion. It may have occured to you that setting a fixed number of

Figure 4.6: Scaling properties of a neural clean-up memory. The number of dimensions required to recover a vector from an input of k bound pairs of vectors, drawn from a lexicon of M vectors is shown. These values are idealizations based on an equation derived by , physical implementations of clean-up using spiking neurons require higher-dimensional vectors to handle equivalently sized lexicons.

dimensions for semantic pointers is simply an mistake. After all, don't we expect that some simple representations may not be of a given dimension, or that some perceptual or motor systems might work with vectors that are of different dimensions? As well, if we have a 'maximum' number of dimensions, will the system simple stop working if we go over that number? There are two kinds of response to these concerns, theoretical and practical. On the theoretical side, we can note that any vectors with lower than the maximum number of dimensions can simply be 'padded' to the appropriate size. In mathematical terms, we can note that vector spaces with smaller numbers of dimensions are subspaces of vector spaces with more dimensions. There is nothing problematic about embedding a lower-dimensional space in a higher-dimensional one. In the opposite case, when there are more dimensions than in the vector space, we can also simply truncate the vector representation. There are more sophisticated ways of 'fitting' higher-dimensional spaces into lower dimensional ones (such as performing singular-value decomposition), but it has been shown that for high-dimensional spaces, simple random mappings into the lower-dimensional space preserve the structure of the higher-dimensional space very well (Bingham and Mannila, 2001). In short, all or most of the structure of other dimensional spaces can be preserved in a high-dimensional space like we have chosen for the SPA. Vector spaces are theoretically quite robust to changes in dimension.

On the more practical side, we can run explicit simulations in which we damage the simulation (thereby not representing the vector space well, or losing dimensions), or add significant amounts of noise to distort the vector space, or again eliminate some dimensions. Performing these manipulations on the binding network described earlier (specifically, the third example in figure 4.4) demonstrates significant robustness to changes in the vector space. For instance, after randomly removing neurons from the central neural group in the network, accurate performance occurred even with an average of 1,221 out of 4000 neurons removed.[5] Similarly, the network is accurate with up to 42% Gaussian noise used to randomly vary the connection weights.[6] Random variation of the weights can be thought to reflect a combination of imprecision in weight maintenance in the synapse, as well as random jitter in incoming spikes. Consequently such simulations speak to both concerns about choosing a specific number of dimensions, and the robustness of

[5]The variance of the simulations was 600 neurons, and both the variance and mean were determined over 500 simulations.

[6]Each synaptic connection weight was independently scaled by a factor chosen from a Gaussian distribution with a mean of 1 and some variance (σ^2). The model continued to function correctly up to σ^2=0.42.

the system to expected neural variability.

Overall, the components of the SPA allow for graceful degredation of performance. This should not be too surprising. After all, previous considerations of neural representation make it clear that there is a smooth link between the number of neurons and the quality of the representation. As well, we have just seen that there is a smooth relationship between the quality of a clean-up memory, the number of dimensions, and the complexity of the representation for a fixed number of neurons. Finally, there are similarly smooth relationships between dimensionality and complexity in ideal VSAs (Plate, 2003).

- constructing clean-ups as context changes, etc. (don't need 60K items necessarily all the time!).
- timing of clean-up memory.
- Adding to/removing items from clean-up? Lots of different clean-up...task/context relevant, general (long-term), etc
- definitely be a sparse representation!

4.6 Example: Fluid intelligence

In cognitive psychology, "fluid intelligence" is distinguished from "crystallized intelligence" to mark the difference between cognitive behavior that depends on currently available information that must be manipulated in some sophisticated manner, and cognitive behavior that depends on potentially distant past experiences. Fluid intelligence is exemplified by solving a difficult planning problem, whereas crystallized intelligence is exemplified by recalling important facts about the world. Most tests that have been designed to measure general intelligence, focus on fluid intelligence. One of the most widely used and respected tools for this purpose is the Raven's Progressive Matrices (RPM; refs?).

In the RPM, subjects are presented with a 3 x 3 matrix, in which each cell in the matrix contains various geometrical figures with the exception of the final cell, which is blank (figure 4.7). The task is to determine which of eight possible answers most appropriately belongs in the blank cell. In order to solve this task, subjects must examine the contents of all of the other cells in the matrix to determine what kind of pattern is there, and then use that pattern to pick the best answer. This, of course, is an excellent example of induction, which I discussed in section 4.4.

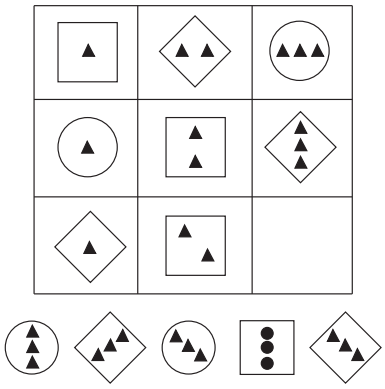


Figure 4.7: Example RPM. Patterns are established in the first two rows of the matrix (see text). Knowledge of these patterns is then tested by requiring the participant to complete the final cell using the options given below the matrix.

While the RPM is an extremely widely used clinical tests, and many experimental examinations have been made of the test, our explicit theoretical understanding of the processes underlying human performance on this task is quite minimal. In fact, to the best of my knowledge, there have been no cognitive models that include the inductive process of rule generation as seen in the RPM. The best-known model of RPM includes all of the possible rules that the system may need to solve the task (ref? carpenter). In solving the task, the model chooses from the available rules, and applies them to the given matrix. However, this treats the RPM like a problem that employs crystallized intelligence, which contradicts its acceptance as a test of fluid intelligence (refs? (Perfetti et al., 2009; Prabhakaran et al., 1997; Gray et al., 2003).).

Recently, however, a graduate student in my lab named Daniel Rasmussen proposed a model of the RPM that is implemented in a spiking neural network (ref?). To understand how his model works, we can consider the example Raven's-style matrix shown in figure 4.7.[7] To correctly solve this problem, a subject needs to extract information from the filled-in cells that identifies three rules: 1) the number of triangles increases by one across the row; 2) the orientation of the triangles stays the same across the row; and 3) each cell also contains a shape in the background, which, across the row includes each of a circle, a square, and a diamond. This combination of rules identifies the correct response (i.e., three triangles tilted

[7]For copyright reasons figure 4.7 shows an example matrix that would not be found on the actual test.

Figure 4.8: Simple RPM example from dan

to the left over top of a circle), which can be determined by applying those rules to the last row. While not all subjects will explicitly formulate these rules, they must somehow extract equivalent information (or get lucky) to get this matrix correct.

In Rasmussen's graduate work, he argues that there are three kinds of rules needed to solve the entire set of problems on the RPM (ref?). The first kind are induction rules, which generalize across neighboring cells of the network (e.g., increase the number of triangles by one). The second kind are set completion rules, which account for set completion across an entire row (e.g., each of a circle, a square, and a diamond are represented within a row). The third kind are visually based XOR rules, which account for the presence and absence of particular visual features across a row. In addition, the choice and coordination of the application of these rules is accounted for by an executive system. For simplicity, here I will only focus on the induction rules (for discussion of other these aspects of the model, see refs?).

In section 4.4, I have already described how the induction rule works for an abstract presented structure. To make this characterization more concrete, consider the simplified Raven's-style matrix shown in figure 4.8. Each cell in this matrix is represented by a semantic pointer which consists of attribute/value pairs. For example, the top right cell could be represented as:[8]

$$\textbf{cell}=\textbf{shape}\otimes\textbf{circle}+\textbf{number}\otimes\textbf{three}.$$

Once all of the cells are represented, the previously described induction rule is provided the pairwise examples in the order they tend to be read by subjects (i.e., across rows, and down columns; ref?carpenter). As the examples are presented, the inferred transformation is built up out of the average of each of the pairwise transformations. That average transformation can then be applied to the second last cell in the matrix. This results in a vector which is compared to each of the possible answers, and the most similar answer is chosen as the correct one by the model.

[8] Surprisingly few decisions need to be made about which attribute value pairs to include in the representations of cells. As long as the information necessary to identify the required transformation is available, the induction rule is generally able to extract it. Thus, including additional information like $\textbf{color}\otimes\textbf{black}+\textbf{orientation}\otimes\textbf{horizontal}+\textbf{shading}\otimes\textbf{solid}$, etc. in this instance does not affect performance.

Figure 4.9: Broader application of the rule induced from ref:previous figure a) applied to 4 triangles b) applied to 4 squares] dan...

Comparing the model across many runs to average human performance on all of the matrices that include an induction rule (x/32? examples), humans score 63% on average, and the model scores 64% on average (ref??? Dan will get these results to me, needs to add a few more sequence matrices to get them all in the test set???). Furthermore, this model can account for several other interesting experimental observations on the RPM. For instance, subjects improve with practice if given the RPM multiple times (ref? Bors, 2003), and also show learning within the span of a single test (Verguts & De Boeck, 2002, ref?). As well, a given subject's performance is not deterministic; given the same test multiple times, subjects will get previously correct answers wrong and vice versa (ref? Bors, 2003). In addition, there are both qualitative and quantitative differences in individual ability; there is variability in "processing power" (variously attributed to working memory, attention, learning ability, or executive functions), but there are also consistent differences in high-level problem-solving strategy between low-scoring and high-scoring individuals (Vigneau et al., 2006, ref?). This is not an exhaustive list, but it represents some of the features that best define human performance, and all of which are captured by the full RPM model (ref???).

That being said, the theoretically most important feature of the model is its ability to successfully perform induction and extract rules. However, simply choosing the best of the eight possible answers might not convince us of that. Much more convincing, is determining how the learned transformation applies in circumstances never encountered by the model. Figure 4.9 shows two examples of this kind of application. In the first instance, the model is asked to apply the learned transform to four triangles. As is evident from figure 4.9a, the preferred answer corresponds to five triangles. This suggests the rule has found the correct 'increase by one' aspect of the rule, at least in the context of triangles. Figure 4.9b shows that, in fact, the 'increase by one' is generic across objects. In this case, the rule is applied to four squares, and the preferred result is five squares. In short, the model has learned the appropriate counting rule, and generalized across objects.

This is a simple, but clear, example of what is sometimes called 'syntactic generalization'. Syntactic generalization is content insensitive generalize; i.e., driven by the syntactic structure of the examples. It is important to note that precisely this kind of generalization has been often been claimed to distinguish cognitive from non-cognitive systems {{refs? 799 Fodor,Jerry 1988; 1241 Jackendoff,R.

2002; 2792 Hummel, John E Holyoak,Keith J. 2003; }}. In this case, it is clear that the generalization is content insensitive because 'increase by one' is applied for squares, triangles, or what have you. I provide more examples of syntactic geneararalization in section 6.5.2.

It might not be too surprising that in order to perform cognitively interesting induction, one has to perform syntactic induction. It may be surprising that once we have an appropriate representational system, the method for performing such syntactic induction is quite simple (i.e. averaging over examples), and can be built into a biologically realistic network. In addition, the same basic architecture is able to capture the other two kinds of reasoning usually identified in cognition: abduction and deduction. In previous work, I showed that the approach underlying the SPA can be used to capture central features of human deductive reasoning (Eliasmith, 2004); I describe this work in more detail in section 6.5.2. Litt and Thagard (ref?) also showed that a version of this model can be used to capture abductive (i.e., inference to the best explanation) reasoning.

This unification of kinds of reasoning in a simple neural mechanism is appealing because it brings together a wide range of what might otherwise seem very different cognitive strategies. Furthermore, having a consistent architecture in which to explore these mechanisms in a biologically constrained manner provides a useful way to begin constructing models of more sophisticated behavior. Of course, I don't take this kind of mechanism to be able to stand on its own as a explanation in the present, simple instantiation. I do think, however, that it provides fertile ground for building more sophisticated models. Ground that is more fertile than past, more symbolically driven approaches.

I take it that it is no accident that past models of the RPM have not had mechanisms for inferring rules. The mechanisms are not there because there is no obvious way to infer the appropriate cognitive rules given the example rules in a symbolic representation. There is little sense to be made of 'averaging' a series of symbolic rules. And, there is no simple replacement for 'averaging' with another operation. With the SPA, however, averaging past examples of rules becomes both very natural, and very powerful. It is poweful because the SP representation allows whatever changes across examples to become 'noise', and whatever stays consistent to be reinforced. Crucially, this is true for both content and syntactic structure.

One final note, which strikingly distinguishes this SPA implementation from standard connectionist approaches, is that there are no connection weight changes in this model. This is true despite the fact that it learns based on past experience, and is able to successfully generalize. I return to this observation in my discussion

on learning in chapter 6.

4.7 Deep semantics for cognition

Why have I returned to the issue of semantics in a chapter on syntax? It is because of the deep and important relations between these two aspects of cognition, which the semantic pointer architecture would be ill-advised to ignore. As I commented in the previous chapter, my consideration of semantics there did not provide a means for integrating a variety of sources of semantics, such as the semantic information coming from different modalities, from motor systems and from lexical considerations. Now, however, we have seen a method for building structures out of sets of semantic pointers.

Consequently, it is a natural extension of the previous discussion on semantics to suggest that the complex kind of semantic structures found in human cognition can be constructed by combining, say, semantic pointers from different modalities into a more unified representation of a multimodal concept. So, for example, the perceptual features of a "robin" might be represented as something like:

$$\mathbf{robinPercept} = \mathbf{visual} \otimes \mathbf{robinVis} + \mathbf{auditory} \otimes \mathbf{robinAud} + \mathbf{tactile} \otimes \mathbf{robinTouch} + ...$$

We could then take those perceptual features to be integrated into a more lexical representation of the concept 'robin' that could look something like:

$$\mathbf{robin} = \mathbf{perceptual} \otimes \mathbf{robinPercept} + \mathbf{isA} \otimes \mathbf{bird} + \mathbf{timeOfYear} \otimes \mathbf{spring} + ...$$

Importantly, the resulting representation in both cases is also a semantic pointer. After all, circular convolution is a compression operation, so the results of binding two vectors is a compressed representation of the content. In this case, what is being represented is not perceptual features as before, but structure. And, just as with perceptual semantic pointers, structured semantic pointers must also be dereferenced by effectively "running the model backwards". In the case of structure, this means unbinding, and cleaning up the results of that operation. In the case of perceptual semantic pointers, this meant decompressing by seeding the model that had been constructed based on perceptual experience. The only difference, it may seem, is that the perceptual model is hierarchical and clean-up memory is not. But this, we will see shortly, may not be a difference after all.

In any case, we now have, theoretically at least, a means of bringing together shallow and deep semantics by binding them into a single semantic pointer. But,

we can go further. One additional, interesting feature of the preceding characterization of the SPA is that transformations of semantic pointers are themselves semantic pointers. For example, when the model solving the Ravens matrices induced a transformation to explain the changes in the structures it had been presented with, the result was a transformation vector that we can decode. That is, it is equivalent to a sum of bound semantic pointers, i.e. a semantic pointer (???make sure this is done above). In some ways, this should not be too surprising given the preceding discussion of the motor control system. After all, in that context semantic pointers were used to drive motor control – a clear case of transformations.

Consequently, since semantic pointers can act as transformations, and conceptual structure can be built up out of bound semantic pointers, our conceptual representations can include characterizations of transformations in them. That is, not only static properties, but also the kinds of changes associated with objects can be encoded in (or pointed to by) the lexical representation of that object. More precisely, I would expect that a compressed description of an object's possible transformations is encoded in the lexical representation of that object. As with any 'part' of such a representation, it would need to be dereferenced, likely by another part of the brain, to decode its semantics. Nevertheless, I believe this is an intriguing and natural way to include an element of time in our understanding of lexical conceptual representations, an element which is often missing in discussions of concepts.

So, I have been suggesting that our conceptual structures can be very complex, including the sum of bound representations of properties, perceptual features, lexical relationships, dynamics, and so on. But, there seems to be an important problem lurking here. Namely, that in my previous description of a clean up memory, I noted that we could accurately decode about 8 slot-filler pairs in a clean up memory that was anatomically plausible. Certainly, more than 8 slots will be needed in order to include all of the kinds of information I have been discussing to this point. How then are we going to be able to encode sophisticated concepts in a biologically realistic manner using the resources of the SPA?

I would like to suggest a solution to this problem, that turns out to allow the architecture to scale extremely well. Namely, rather than constraining ourselves to a single memory for encoding capacity, we can chain memories together, and realize a far superior scaling. In short, I am suggesting that we can decode in one memory, and take the decoded vectors to another memory that we know can decode it. At each stage of such a chain the vector to be decoded comes from the cleanup of the previous stage, ensuring full capacity of the current stage is available for further decoding the vector. As a result, the effective capacity will

scale as the power of the number of stages in such a chain, but each stage will only require a linear increase in the number of neurons. Let us consider a concrete example.

Consider a simple example of how the 'dog' semantic pointer might be constructed. It might include slot and filler pairs for visual information, auditory information, other modalities, as well as various lexical relationships. Suppose for this simple example that there are only three slots for lexical information. The representation of this pointer might then look something like:

$$\mathbf{dog} = \mathbf{isA} \otimes \mathbf{mammal} + \mathbf{hasProperty} \otimes \mathbf{friendly} + \mathbf{likes} \otimes \mathbf{bones}$$

Suppose that with 100 dimensional semantic space, we can decode any elements in a three slot representation with 99.9% accuracy. If we query our 'dog' concept, wanting to know what it is (i.e., determining the filler in the 'isA' slot) we will discover that a dog is a mammal. That is, the result of this operation will be a clean version of the 'mammal' semantic pointer.

We might then decode this new semantic pointer with the second memory in our chain. Suppose that this semantic pointer is the sum of three other slot filler pairs, perhaps

$$\mathbf{mammal} = \mathbf{isA} \otimes \mathbf{animal} + \mathbf{hasProperty} \otimes \mathbf{hair} + \mathbf{example} \otimes \mathbf{human}$$

We could then decode any of the values in those slots also with 99.9% accuracy. That is, we could determine that a mammal is an animal, for instance. And so the chaining might continue with the results of that decoding. What is interesting to note here is that in the second memory we could have decoded any three slot-filler pairs from any of the three slot-filler pairs in the memory earlier in the chain. So, effectively, our original concept can include nine slot-filler pairs that we would be able to decode with near 100% accuracy (the expected accuracy is 99.9%*99.9%=99.8%).

In short, we can increase the effective number of slot-filler pairs by chaining the space. As figure 4.10 shows, the scaling of a chained space significantly outperforms an un-chained space. It is clear from this figure that even with the same total number of dimensions, an un-chained space can not effectively decode very many pairs (the number of pairs is approximately linear in the number of dimensions). In contrast, the number of decodable pairs in the chained space is equal to the product of the number of decodable pairs in each space separately. In this example it means we can effectively decode 150 instead of 20 slot-filler pairs with the same number of dimensions.

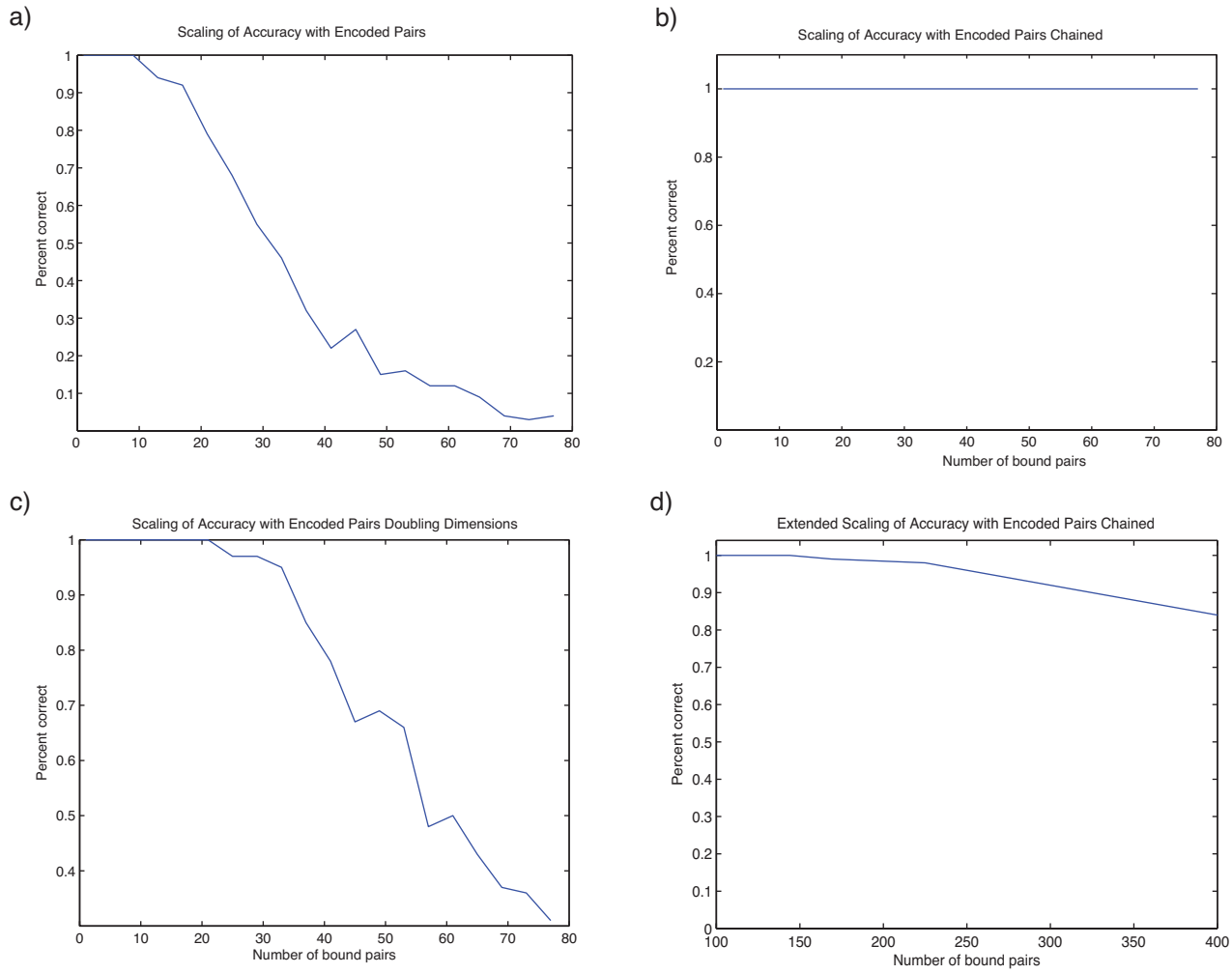


Figure 4.10: Factorization effects for encoding large conceptual structures. A) The accuracy versus number of encoded slot-filler pairs rapidly decreases in a single memory. This is a 550-dimensional space with 60,000 possible terms. The accuracy falls to 3% for 80 pairs. B) The accuracy of a pair of memories that are chained. Each space is 550-dimensional and has 60,000 possible terms. The decoding accuracy is 100% for up to 80 pairs C) The accuracy of a single memory with the same total number of dimensions as the chained space. This is a 1100-dimensional space with 60,000 possible terms. The decoding accuracy is 31% for 80 pairs. D) The extended scaling of the chained space between 100 and 400 pairs. The scaling begins to delince smoothly around 150 pairs, but is still 84% accurate for 400 pairs. All points are averages of 100 runs.

We can project the expected accuracy for much larger sets of pairs by turning to our earlier consideration of clean-up memory. For instance, we know that we can clean-up in a 1000 dimensional space about 8 pairs from a vocabulary of 60,000 terms with 99% accuracy using the number of neurons consistent with observed connectivity (see figure 4.6). If we chain 4 such memories together, we would be able to decode $8^4 = 4096$ pairs with $.99^4 = .96$, or 96%, accuracy. This kind of scaling makes earlier concerns about encoding sophisticated concepts much less pressing.

Why do we get such good scaling? In short, it is because we have introduced a nonlinearity (the clean-up) into our decoding process that is not available in a single memory. That nonlinearity acts to align the output of one memory with the known values in the next semantic space in the chain. It may seem that this enforces a kind of hierarchy on our conceptual structure, but it does not. The reason I use the word 'chain' instead of 'hierarchy' is because there are many kinds of chains that could be implemented in this way, only some of which are hierarchies. For instance, the second memory in a two-level chain could be the same as the first (a recursive chain). Or, each possible slot could have its own dedicated memory for decoding items of that slot-type (a strict hierarchy). It is unclear which of these, or other, kinds of structure will be most appropriate for characterizing human-like semantic spaces. And, mapping this chained structure carefully to data on human concepts is beyond the scope of this book, despite its clear importance.[9] It is intriguing, however, that a hierarchy is a very natural kind of chain to implement, and that there is strong evidence for this kind of structure to human concepts (ref???).

Nevertheless, I believe these chained semantic spaces provide a crucial demonstration of how syntax and semantics can be used together to represent very large, intricate conceptual spaces in ways we know can be plausibly be implemented in the brain. These semantic spaces are tied directly to perceptual and motor experiences as described in the previous chapter, and can also be used to encode language-like structures that capture lexical relationships. Furthermore, we can transform these representations based on their structure, and also learn to extract transformations that generalize over the encoded syntax. The stage is now set for considering how to build a system which itself uses, transforms, and generates such representations.

[9]Rogers and McLelland (ref???) is an excellent example of tackling this difficult task in a constructive way.

4.8 Nengo: Structured representations in neurons

- Theoretical point: Binding and summing for structured repn
- do tutorial on... just convolution and summing yes. how much to download? mathematical details in an appendix?