Holographic Reduced Representations

Tony Plate
Department of Computer Science, University of Toronto
Toronto, Ontario, Canada, M5S 1A4
tap@ai.utoronto.ca

Abstract

Associative memories are conventionally used to represent data with very simple structure: sets of pairs of vectors. This paper describes a method for representing more complex compositional structure in distributed representations. The method uses circular convolution to associate items, which are represented by vectors. Arbitrary variable bindings, short sequences of various lengths, simple frame-like structures, and reduced representations can be represented in a fixed width vector. These representations are items in their own right, and can be used in constructing compositional structures. The noisy reconstructions extracted from convolution memories can be cleaned up by using a separate associative memory that has good reconstructive properties.

I Introduction

Distributed representations [13] are attractive for a number of reasons. They offer the possibility of representing concepts in a continuous space, they degrade gracefully with noise, and they can be processed in a parallel network of simple processing elements. However, the problem of representing compositional structure[1] in distributed representations has been for some time a prominent concern of both proponents and critics of connectionism [9, 32, 12].

Most work on neural-network style associative memories has focussed on either auto-associative or hetero-associative memories. Auto-associative memories, e.g., Hopfield networks [14], store an unordered set of items. They can be used to recall item given a distorted version of one. Hetero-associative memories, e.g., holographic memories and matrix memories [37, 8, 22, 5, 38], store a set of pairs of items. One item of a pair can be recalled using the other as a cue. Matrix style memories are the more popular class, owing to superior storage capacity and fewer constraints on vectors to be stored.

For artificial intelligence tasks such as language processing and reasoning the need arises to represent more complex data structures such as sequences and trees. It is difficult to represent sequences or trees in distributed representations using associations of pairs (or even n-tuples) of items and retain the benefits of distributed representations. The problem with representing compositional structure in most associative memories is that items and associations are represented in different spaces. For example, in a Hopfield memory (a matrix style memory) items are represented on unit activations (a vector) and associations are represented on connections weights (a matrix). This makes it difficult to represent relationships with recursive structure in which an association of items may be the subject of another association.

Hinton [12] discusses this problem and proposes a framework in which "reduced descriptions" are used to represent parts and objects in a part-whole hierarchy (a frame-like representation). This framework requires that a number of vectors, each a part and together forming a whole, be compressed (reduced) into a single vector of the same dimension as the original vectors. This reduced vector can in turn be used as a part in the representation of some greater whole. The reduction must be reversible so that one can move in both directions in a part-whole hierarchy, i.e., reduce a set of vectors (a whole) to a single vector (a potential part), and expand a single vector (a part) to a set of vectors (a whole). In this way, compositional structure is represented. An essential aspect of reduced descriptions is that they should be systematically related to their components, so that information about the components can be gleaned without expansion. It is this aspect that distinguishes reduced descriptions from arbitrary pointers. Unfortunately, Hinton does not suggest any concrete way of performing the reduction and expansion mappings.

Some researchers have built models or designed frameworks in which some compositional structure is present in distributed representations. For some examples see the papers of Touretzky [33], Pollack [27], or Smolensky [32].

In this paper I propose a new method for representing compositional structure in distributed representations. Circular convolutions are used to construct associations of vectors. The representation of an association is a vector of the same dimensionality as the vectors which are associated. This allows the construction of representa-

[1] I.e., recursive, or tree-like structure.

IEEE Transactions on Neural Networks 6:3:623-641 1995

tions of objects with compositional structure. I call these *Holographic Reduced Representations* (HRRs), since convolution and correlation based memories are closely related to holographic storage, and they provide an implementation of Hinton's [12] reduced descriptions. I describe how HRRs and auto-associative item memories can be used to build distributed connectionist systems which manipulate complex structures. The item memories are necessary to clean up the noisy items extracted from the convolution representations.

Convolution/correlation (holographic) memories have been generally regarded as inferior to matrix style associative memories for associating pairs of items, for reasons concerning capacity and constraints (see [37, 8]). However, matrix style memories have a problem of expanding dimensionality when used for representing compositional structure. Convolution/correlation memories do not have this problem. Their storage capacity is sufficient to be useful and restrictions on the classes of vectors can be coped with by using matrix style associative memories to transform unsuitable vectors.

Associative memories are reviewed in Section II. A interpretation of convolution as a compressed outer product is given. In Section III addition memories are reviewed. The need for high capacity error-correcting associative memories is discussed in Section IV. Representations of more complex structures are discussed in Section V; some sequence representations are reviewed and ways of doing variable binding and representing simple frame structures are suggested. The idea of Holographic Reduced Representations (HRRs) falls naturally out of these representations, and is discussed in Section V.E. In Section VI I discuss issues of representing features and tokens with the types of vectors that convolution memories can work with. I describe two simple machines that use HRRs in Section VII. Various mathematical properties are discussed in Section VIII, including the relationship between convolution and fast Fourier transforms and the status of correlation as an approximate inverse to convolution. The capacity of convolution memories and HRRs are discussed in Section IX. In Section X I give examples of the construction and decoding of HRRs.

II Hetero-associative memory – matrix and convolution implementations

Hetero-associative memories are used to store associations between pairs of vectors. The vectors are usually distributed representations of discrete items (e.g., (images).

Convolution-correlation memories (sometimes referred to as holographic-like memories) and matrix memories have been regarded as alternative methods for implementing hetero-associative memory [37, 19, 23, 30, 8]. Matrix memories have received more interest, due to their relative simplicity, their higher capacity in terms of the dimensionality of the vectors being associated, and their relative lack of constraints on those vectors.

II.A Associative memories

There are three operations used in most non-adaptive associative memories: encoding, decoding, and trace composition. The encoding operation takes two item vectors and produces a memory trace (a vector or a matrix). The decoding operation takes a memory trace and a single item (the cue), and produces the item that was originally associated with the cue, or a noisy version thereof. Memory traces can be composed by addition or the binary-OR operation. The decoding operation will work with this sum of individual traces, but the retrieved items may be noisier. In some models the encoding and decoding operations are bi-linear, e.g., Murdock [19], in others the decoding operation is non-linear, e.g., Hopfield [14], and in others all the operations are non-linear, e.g., Willshaw [37].

To illustrate this, let $\mathbb{I}$ be the space of vectors representing items, and $\mathbb{T}$ be the space of vectors or matrices representing memory traces. There are often constraints on the vectors, e.g., they should be nearly orthogonal. Let

$$\boxtimes : \mathbb{I} \times \mathbb{I} \rightarrow \mathbb{T}$$

be the encoding operation,

$$\rhd : \mathbb{I} \times \mathbb{T} \rightarrow \mathbb{I}$$

be the decoding operation, and

$$\boxplus : \mathbb{T} \times \mathbb{T} \rightarrow \mathbb{T}$$

be the trace composition operation. Let $\tilde{\mathbf{a}}$, $\tilde{\mathbf{b}}$, $\tilde{\mathbf{c}}$, $\tilde{\mathbf{d}}$, $\tilde{\mathbf{e}}$, and $\tilde{\mathbf{f}}$ be item vectors, and let T_i be memory traces.

The association of two items $\tilde{\mathbf{a}}$ and $\tilde{\mathbf{b}}$ is represented by the trace

$$T_1 = \tilde{\mathbf{a}} \boxtimes \tilde{\mathbf{b}}.$$

We can recover $\tilde{\mathbf{b}}$ from T_1 by using the decoding operation on T_1 and the cue $\tilde{\mathbf{a}}$:

$$\tilde{\mathbf{a}} \rhd T_1$$

gives $\tilde{\mathbf{b}}$, or a noisy version of it. Noisy versions of $\tilde{\mathbf{a}}$ can also be used as cues and, depending on the properties of the particular scheme, the retrieved vector will be more or less similar to $\tilde{\mathbf{b}}$.

A trace can represent a number of associations, e.g.,

$$T_2 = (\tilde{\mathbf{a}} \boxtimes \tilde{\mathbf{b}}) \boxplus (\tilde{\mathbf{c}} \boxtimes \tilde{\mathbf{d}}) \boxplus (\tilde{\mathbf{e}} \boxtimes \tilde{\mathbf{f}}).$$

The first item from any pair[2] can be used as a cue to recover the other item of the pair, e.g.,

$$\tilde{\mathbf{c}} \rhd T_2$$

gives a noisy version of $\tilde{\mathbf{d}}$. The noisiness of the recovered vector increases with the number of associations stored in a single memory trace. The number of associations that can be represented usefully in a single trace is usually referred to as the *capacity* of the memory model.

[2] Matrix memories are usually not symmetric; to use $\tilde{\mathbf{d}}$ as a cue T_2 must be transposed. Convolution memories are symmetric: either member of the pair can be used as a cue.

In matrix memories the encoding operation is the outer product, and in convolution memories the encoding operation is convolution. Addition and the binary-OR operation have both been used as the trace composition operation in matrix and convolution memories.

Both matrix and convolution memories, especially the versions with linear encoding operations, have the property that they preserve similarity. That is, if items $\tilde{\mathbf{a}}$ and $\tilde{\mathbf{a}}'$ are similar, and items $\tilde{\mathbf{b}}$ and $\tilde{\mathbf{b}}'$ are similar, then the traces $\tilde{\mathbf{a}} \boxtimes \tilde{\mathbf{b}}$ and $\tilde{\mathbf{a}}' \boxtimes \tilde{\mathbf{b}}'$ will also be similar. The degree of similarity of the traces will be related to the degree of similarity of the items. This property is potentially very useful because it allows an estimate of the similarity of traces to be computed without decoding.

II.B The problem of complex structure

Pairwise associations do not suffice for the practical representation of more complex data structures, such as trees. The need to represent such data structures arises in systems which use higher-order predicates, e.g., predicates such as "cause", "think", and "believe", in language processing or reasoning systems.

One approach to representing more complex data structures in associative memory is to use three-way associations, as are used in LISP data structures (car, cdr, and address). Touretzky and Hinton [35] and Touretzky [33] describe systems based on this idea. A major problem with this approach is that access is slow; many pointers must be followed to determine the constituents of a structure. This removes one of the major advantages of distributed representations; fast determination of similarity.

Another approach is to use an associative memory operator that can be applied recursively. This corresponds to an operator that can map from $\mathbb{I} \times \mathbb{T} \rightarrow \mathbb{T}'$, and $\mathbb{I} \times \mathbb{T}' \rightarrow \mathbb{T}''$, etc. A major problem with most implementations of this approach is the expanding dimensionality of the association spaces $\mathbb{T}$, $\mathbb{T}'$, $\mathbb{T}''$, etc. Vectors that grow arbitrarily in dimension are difficult to use in practical systems. This approach has been used by a number of researchers, and the problem of expanding dimensionality has been tackled in a number of ways. Metcalfe Eich [18] and Murdock [20] both describe methods based on aperiodic convolution. Metcalfe Eich discards outside elements of convolution products to avoid expanding dimensionality. Murdock uses infinite-dimensional vectors. Smolensky [32] proposes Tensor product memories, which use a generalized outer product as the associative operator. In these memories the dimensionality of the association space is exponential in the depth of recursion involved. Smolensky suggests placing a hard limit on the depth of recursion in order to keep the size of the association space tractable (e.g., no structure can be more than 4 levels deep). In a later paper Legendre, Miyata, Smolensky [16] describe a scheme which permits a soft limit on the depth of recursion, though its properties as the limit is approached or exceeded are not clear. In Pollack's [27] Recursive Auto-Associative Memories (RAAMs) items, associations, and recursive associations are all represented in the same vector space. A back-propagation network learns the encoding and decoding mappings. This solves the problem of expanding dimensionality. However, the learning is slow and the generalization of the mappings to novel items and structures is highly variable. In HRRs items and associations are also represented in the same vector space and circular convolution and its approximate inverse are used as the encoding and decoding operators.

II.C Convolution-correlation memories

In nearly all convolution memory models the aperiodic convolution operation has been used to form associations.[3] Traces are usually composed by addition. The aperiodic convolution of two vectors each with n elements results in a vector with $2n-1$ elements. This result can be convolved with another vector (recursive convolution); and if that vector has n elements, the result has $3n-2$ elements. Thus the dimensionality of the resulting vectors expands with recursive convolution.[4]

The problem of expanding dimensionality can be avoided entirely by the use of circular convolution, an operation well known in signal processing (e.g., see [10]). The result of the circular convolution of two vectors of n elements has just n elements.

Matrix and convolution memories provide different instantiations of the abstract associative memory operators set out in Section II.A. However, they are more closely related than might be suggested by this. The convolution of two vectors (whether circular or aperiodic) can be regarded as a compression of the outer product of those two vectors. The compression is achieved by summing along the top-right to bottom-left diagonals of the outer product, as illustrated in Figures 1-5.

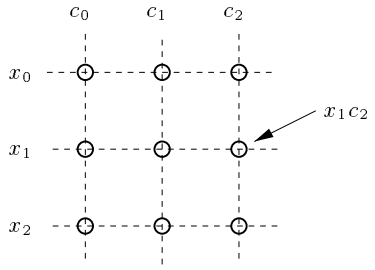
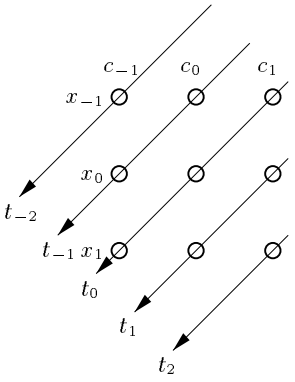


Figure 1: The outer product of two vectors, $\tilde{\mathbf{c}}$ and $\tilde{\mathbf{x}}$ with the content of an example location shown.

The outer product of two vectors is illustrated in Figure 1, which is intended to help with the understanding of the four subsequent figures. Figure 2shows standard aperiodic convolution, and Figure 3 shows the truncated aperiodic convolution used by Metcalfe Eich [18]. The

[3] The exception is the non-linear correlograph of Willshaw [37], first published in 1969.

[4] For the sake of mathematical elegance, many authors have considered the vectors to have an infinite number of elements centered on the zero'th element, i.e., indexed from $-\infty$ through 0 to ∞. The vectors must have a finite number of non-zero elements in order for the convolution operation to be defined, and these are usually centered about the zero'th element [20, 3, 26].



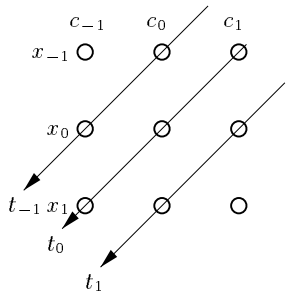
$$\tilde{\mathbf{t}} = \tilde{\mathbf{c}} * \tilde{\mathbf{x}}$$

$$\begin{aligned} t_{-2} &= c_{-1}x_{-1} \\ t_{-1} &= c_{-1}x_0 + c_0x_{-1} \\ t_0 &= c_{-1}x_1 + c_0x_0 + c_1x_{-1} \\ t_1 &= c_0x_1 + c_1x_0 \\ t_2 &= c_1x_1 \end{aligned}$$

$$t_j = \sum_{k=-(n-1)/2}^{(n-1)/2} c_k x_{j-k}$$

for $j = -(n-1)$ to $n-1$

Figure 2: Aperiodic convolution represented as a compressed outer product for $n = 3$. The indices are centered on zero since vectors "grow" (at both ends) in dimensionality with repeated convolutions.



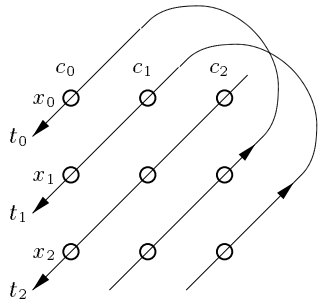
$$\tilde{\mathbf{t}} = \tilde{\mathbf{c}} *_t \tilde{\mathbf{x}}$$

$$\begin{aligned} t_{-1} &= c_{-1}x_0 + c_0x_{-1} \\ t_0 &= c_{-1}x_1 + c_0x_0 + c_1x_{-1} \\ t_1 &= c_0x_1 + c_1x_0 \end{aligned}$$

$$t_j = \sum_{k=-(n-1)/2}^{(n-1)/2} c_k x_{j-k}$$

for $j = -\frac{n-1}{2}$ to $\frac{n-1}{2}$

Figure 3: Metcalfe's truncated aperiodic convolution represented as a compressed outer product for $n = 3$.



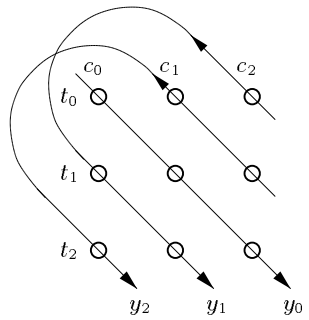
$$\tilde{\mathbf{t}} = \tilde{\mathbf{c}} \circledast \tilde{\mathbf{x}}$$

$$\begin{aligned} t_0 &= c_0x_0 + c_2x_1 + c_1x_2 \\ t_1 &= c_1x_0 + c_0x_1 + c_2x_2 \\ t_2 &= c_2x_0 + c_1x_1 + c_0x_2 \end{aligned}$$

$$t_j = \sum_{k=0}^{n-1} c_k x_{j-k}$$

for $j = 0$ to $n-1$
(Subscripts are modulo-n)

Figure 4: Circular convolution represented as a compressed outer product for $n = 3$.



$$\tilde{\mathbf{y}} = \tilde{\mathbf{c}} \circledcirc \tilde{\mathbf{t}}$$

$$\begin{aligned} y_0 &= c_0t_0 + c_1t_1 + c_2t_2 \\ y_1 &= c_2t_0 + c_0t_1 + c_1t_2 \\ y_2 &= c_1t_0 + c_2t_1 + c_0t_2 \end{aligned}$$

$$y_j = \sum_{k=0}^{n-1} c_k t_{k+j}$$

for $j = 0$ to $n-1$
(Subscripts are modulo-n)

Figure 5: Circular correlation represented as a compressed outer product for $n = 3$.

circular convolution operation, $\circledast$, is illustrated in Figure 4. Elements are summed along the indicated transdiagonals in these figures. While the circular convolution operation is straightforward, what is remarkable is that circular correlation, $\circledcirc$, (illustrated in Figure 5) is an approximate inverse operation of it.[5] If a pair of vectors is convolved together to give a memory trace, then one member of the pair can be correlated with the trace to produce the other member of the pair. Suppose we have a trace which is the convolution of a cue with another vector, $\tilde{\mathbf{t}} = \tilde{\mathbf{c}} \circledast \tilde{\mathbf{x}}$. Then correlation allows the reconstruction of a distorted version of $\tilde{\mathbf{x}}$ from $\tilde{\mathbf{t}}$ and $\tilde{\mathbf{c}}$: $\tilde{\mathbf{y}} = \tilde{\mathbf{c}} \circledcirc \tilde{\mathbf{t}}$ and $\tilde{\mathbf{y}} \approx \tilde{\mathbf{x}}$. The correlation operation also has aperiodic and circular versions, which are approximate inverses for the respective convolution operations.

Multiple associations can be represented by the sum of the individual associations. Upon decoding the contribution of the irrelevant terms can be ignored as distortion. For example, if $\tilde{\mathbf{t}} = \tilde{\mathbf{c}}_1 \circledast \tilde{\mathbf{x}}_1 + \tilde{\mathbf{c}}_2 \circledast \tilde{\mathbf{x}}_2$, then the result of decoding of $\tilde{\mathbf{t}}$ with $\tilde{\mathbf{c}}_1$ is $\tilde{\mathbf{c}}_1 \circledcirc \tilde{\mathbf{c}}_1 \circledast \tilde{\mathbf{x}}_1 + \tilde{\mathbf{c}}_1 \circledcirc \tilde{\mathbf{c}}_2 \circledast \tilde{\mathbf{x}}_2$. If the vectors have been chosen randomly the second term will, with high probability, have low correlation with all of $\tilde{\mathbf{c}}_1, \tilde{\mathbf{c}}_2, \tilde{\mathbf{x}}_1$ and $\tilde{\mathbf{x}}_2$ and the sum will be recognizable as a distorted version of $\tilde{\mathbf{x}}_1$.

II.D Distributional constraints on the elements of vectors

A sufficient condition for correlation to decode convolution is that the elements of each vector (of dimension n) be independently and identically distributed with mean zero and variance $1/n$. This results in the expected Euclidean length of a vector being one. Examples of suitable distributions for elements are the normal distribution and the discrete distribution with values equiprobably $\pm 1/\sqrt{n}$. The reasons for these distributional constraints should become apparent in the next sub-section.

The tension between these constraints and the conventional use of particular elements of vectors to represent meaningful features in distributed representations is discussed in Section VI.

II.E Why correlation decodes convolution

It is not immediately obvious why correlation decodes convolution. However, it is not hard to see if an example is worked through. Consider vectors with three elements, $\tilde{\mathbf{c}} = (c_0, c_1, c_2)$, and $\tilde{\mathbf{x}} = (x_0, x_1, x_2)$, where the x_i and c_i are independently drawn from $\mathrm{N}(0, \frac{1}{n})$ (i.e., a normal distribution with mean zero and variance $1/n$, $n = 3$ in this example). The convolution of $\tilde{\mathbf{c}}$ and $\tilde{\mathbf{x}}$ is:

$$\tilde{\mathbf{c}} \circledast \tilde{\mathbf{x}} = \begin{bmatrix} c_0x_0 + c_1x_2 + c_2x_1 \\ c_0x_1 + c_1x_0 + c_2x_2 \\ c_0x_2 + c_1x_1 + c_2x_0 \end{bmatrix}$$

The decoding of this trace with $\tilde{\mathbf{c}}$ to retrieve $\tilde{\mathbf{x}}$ is:

$$\tilde{\mathbf{c}} \circledcirc (\tilde{\mathbf{c}} \circledast \tilde{\mathbf{x}})$$

[5] Provided that the elements of the vectors satisfy certain distributional constraints.

$$= \begin{bmatrix} x_0(c_0^2+c_1^2+c_2^2)+x_1c_0c_2+x_2c_0c_1+x_1c_0c_1 \\ \qquad +x_2c_1c_2+x_1c_1c_2+x_2c_0c_2 \\ x_1(c_0^2+c_1^2+c_2^2)+x_0c_0c_1+x_2c_0c_2+x_0c_0c_2 \\ \qquad +x_2c_1c_2+x_0c_1c_2+x_2c_0c_1 \\ x_2(c_0^2+c_1^2+c_2^2)+x_0c_0c_1+x_1c_1c_2+x_0c_1c_2 \\ \qquad +x_1c_0c_2+x_0c_0c_2+x_1c_0c_1 \end{bmatrix}$$

$$= \begin{bmatrix} x_0(1+\xi)+\eta_0 \\ x_1(1+\xi)+\eta_1 \\ x_2(1+\xi)+\eta_2 \end{bmatrix} = (1+\xi)\tilde{\mathbf{x}}+\tilde{\eta}$$

where ξ and the η_i can be treated as zero mean noise. The variances of ξ and the η_i are inversely proportional to n. The distributions of the ξ and η_i are normal in the limit as n goes to infinity, but the approximation is good for n as small as 16. Typical values for n in convolution associative memory systems are in the hundreds and thousands.

Using the central limit theorem, and assuming the c_i and x_i and independent and distributed as $\mathrm{N}(0, \frac{1}{n})$, the distributions of ξ and the η_i for large n are:

$\xi \stackrel{d}{=} \mathrm{N}(0, \frac{2}{n})$, since $\xi = (c_0^2 + c_1^2 + ... + c_n^2) - 1$, and the c_i^2 are independent and have mean $1/n$ and variance $2/n^2$.

$\eta_i \stackrel{d}{=} \mathrm{N}(0, \frac{n-1}{n^2})$, since the $n(n-1)$ terms like $x_j c_k c_l$ $(k \neq l)$ have mean zero and variance $1/n^3$, and the pairwise covariances of these terms are zero.

It is most useful to calculate the variance of the dot product $\tilde{\mathbf{x}} \cdot (\tilde{\mathbf{c}} \circledast (\tilde{\mathbf{c}} \circledast \tilde{\mathbf{x}}))$ as this gives a measure of the similarity of the original and reconstructed versions of $\tilde{\mathbf{x}}$. However, this requires taking into account the covariances of the noise terms in the different elements. Extensive tables of variances for dot products of various convolution products have been compiled by Weber [36] for aperiodic convolution. Unfortunately, these do not apply exactly to circular convolution. The means and variances for dot products of some common circular convolution products are given in Table 1 in Section VIII.A.

II.F Relationship of convolution to correlation

The correlation of $\tilde{\mathbf{c}}$ and $\tilde{\mathbf{t}}$ is equivalent to the convolution of $\tilde{\mathbf{t}}$ with the *involution*[6] of $\tilde{\mathbf{c}}$. The involution of $\tilde{\mathbf{c}}$ is the vector $\tilde{\mathbf{d}} = \tilde{\mathbf{c}}^*$ such that $d_i = c_{-i}$, where subscripts are modulo-n. For example, if $\tilde{\mathbf{c}} = (c_0, c_1, c_2, c_3)$, then $\tilde{\mathbf{c}}^* = (c_0, c_3, c_2, c_1)$. Writing $\tilde{\mathbf{c}}^* \circledast \tilde{\mathbf{t}}$ is preferable to writing $\tilde{\mathbf{c}} \circledcirc \tilde{\mathbf{t}}$ because it simplifies algebra, since correlation is neither associative nor commutative whereas convolution is both. Furthermore, in an analogy with inverse matrices, it is sometimes convenient to refer to $\tilde{\mathbf{c}}^*$ as the *approximate inverse* of $\tilde{\mathbf{c}}$. The exact inverse of vectors under convolution (i.e., $\tilde{\mathbf{c}}^{-1}$) are discussed in Section VIII.C.

II.G How much information is stored in a convolution trace

Since a convolution trace only has n numbers in it, it may seem strange that several pairs of vectors can be stored in it, since each of those vectors also has n numbers. The reason is that the vectors are stored with very poor fidelity. The convolution trace stores enough information to recognize the vectors in it, but not enough to reconstruct them accurately. To store a vector in a recognition memory we only need to store enough information to discriminate it from the other vectors. If M vectors are used to represent M different (equiprobable) items, then about $2k \log_2 M$ bits of information are needed to represent k pairs of those items for the purposes of recognition.[7] The dimensionality of the vectors does not enter into this calculation, only the number of vectors matters.

For example, if we have 1024 items (each represented by a different vector), then the number of bits required to store three pairs of those items is slightly less than $2 \times 3 \times \log_2 1024 = 60$ bits. A convolution memory using random vectors with 512 elements would comfortably be able to store three pairs. Storing 60 bits of information in 512 floating point numbers is not very efficient, but for the storage of complex structure this is not a critical issue.

III Addition Memories

One of the simplest ways to store a set of vectors is to add them together. Such storage does not allow for recall or reconstruction of the stored items, but it does allow for recognition, i.e., determining whether a particular item has been stored or not. A real-world example of this is the easy recognition of objects in a multiple-exposure photograph. The properties of addition memories determine the characteristics of storage of multiple items in convolution memories.

The principle of addition memory can be stated as "adding together two high dimensional vectors gives a vector which is similar to each and not very similar to anything else."[8] This principle underlies both convolution and matrix memories and the same sort of analysis can be applied to the linear versions of each.

An analysis of the capacity of addition memories is given in Appendix A. Note that it is not necessary for elements of vectors to have continuous values for addition memories to work. Furthermore, their capacity can be improved by applying a suitable non-linear (e.g., threshold) function to the trace. Touretzky and Hinton [35] and Rosenfeld and Touretzky [28] discussed binary-OR memories[9], which can be viewed as a non-linear version of an addition memories. Binary-OR memories were used in the model of Touretzky and Hinton [35].

IV The need for reconstructive item memories

Convolution memories share the inability of addition memories to provide accurate reconstructions. Conse-

[6] Involution has a more general meaning, but in this paper I use it to mean a particular operation.

[7] Actually, slightly less than $2k \log_2 M$ bits are required since the pairs are unordered.

[8] This applies to the degree that the elements of the vectors are randomly and independently distributed.

[9] In a binary-OR memory binary vectors are logically OR'ed together instead of being added.

quently, if a system using convolution representations is to do some sort of recall (as opposed to recognition), it must have an additional error-correcting auto-associative item memory. This is needed to clean up the noisy vectors retrieved from the convolution traces. This reconstructive memory must store all the items that the system can produce. When given as input a noisy version of one of those items it must either output the closest item or indicate that the input is not close enough to any of the stored items. Note that one convolution trace stores only a few associations or items, and the item memory stores many items.

For example, suppose the system is to store pairs of random vectors $\tilde{\mathbf{a}}, \tilde{\mathbf{b}}, ..., \tilde{\mathbf{z}}$. The item memory must store these 26 vectors and must be able to output the closest item for any input vector (the "clean" operation). Such a system is shown in Figure 6. The trace is a sum of convolved pairs, e.g., $\tilde{\mathbf{t}} = \tilde{\mathbf{a}} \circledast \tilde{\mathbf{b}} + \tilde{\mathbf{c}} \circledast \tilde{\mathbf{d}} + \tilde{\mathbf{e}} \circledast \tilde{\mathbf{f}}$. The system is given one item as an input cue and its task is to output the item that cue was associated with in the trace. It should also output a scalar value (the strength) which is high when the input cue was a member of a pair, and low when the input cue was not a member of a pair. When given the above trace $\tilde{\mathbf{t}}$ and $\tilde{\mathbf{a}}$ as a cue it should produce $\tilde{\mathbf{b}}$ and a high strength. When given $\tilde{\mathbf{g}}$ as a cue it should give a low strength. The item it outputs is unimportant when the strength is low.

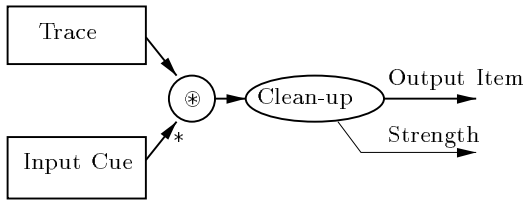


Figure 6: A hetero-associator machine. The "*" on the operand to the convolution indicates the approximate inverse is taken.

The exact method of implementation of the item memory is unimportant. Hopfield networks are probably not a good candidate because of their low capacity in terms of the dimension of the vectors being stored. Kanerva networks [15] have sufficient capacity, but can only store binary vectors.[10] For the simulations reported in Appendix B. I stored vectors in an array and computed all dot-products in order to find the closest match.

V Representing more complex structure

Pairs of items are easy to represent in many types of associative memory, but convolution memory is also suited to the representation of more complex structure.

V.A Sequences

Sequences can be represented in a number of ways using convolution encoding. An entire sequence can be represented in one memory trace, with the probability of error increasing with the length of the stored sequence. Alternatively, chunking can be used to represent a sequence of any length in a number of memory traces.

Murdock [19, 21], and Lewandowsky and Murdock [17] propose a chaining method of representing sequences in a single memory trace, and model a large number of psychological phenomena with it. The technique used stores both item and pair information in the memory trace, for example, if the sequence of vectors to be stored is $\tilde{\mathbf{a}}\tilde{\mathbf{b}}\tilde{\mathbf{c}}$, then the trace is

$$\alpha_1 \tilde{\mathbf{a}} + \beta_1 \tilde{\mathbf{a}} \circledast \tilde{\mathbf{b}} + \alpha_2 \tilde{\mathbf{b}} + \beta_2 \tilde{\mathbf{b}} \circledast \tilde{\mathbf{c}} + \alpha_3 \tilde{\mathbf{c}},$$

where the α_i and β_i are weighting parameters, with $\alpha_i > \alpha_{i+1}$. The retrieval of the sequence begins with retrieving the strongest component of the trace, which will be $\tilde{\mathbf{a}}$. From there the retrieval is by chaining — correlating the trace with the current item to retrieve the next item. The end of the sequence is detected when the correlation of the trace with the current item is not similar to any item in the item memory. This representation of sequences has the properties that a sequence is similar to all of the items in it, retrieval can start from any given element of the sequence, and similar sequences will have similar representations. It has the disadvantage that some sequences with repeated items cannot be properly represented.

Another way to represent sequences is to use the entire previous sequence as context rather than just the previous item [21]. This makes it possible to store sequences with repeated of items. To store $\tilde{\mathbf{a}}\tilde{\mathbf{b}}\tilde{\mathbf{c}}$, the trace is:

$$\tilde{\mathbf{a}} + \tilde{\mathbf{a}} \circledast \tilde{\mathbf{b}} + \tilde{\mathbf{a}} \circledast \tilde{\mathbf{b}} \circledast \tilde{\mathbf{c}}.$$

This type of sequence can be retrieved in a similar way to the previous, except that the retrieval cue must be built up using convolutions.

The retrieval of later items in both these representations could be improved by subtracting off prefix components as the items in the sequence are retrieved.

Yet another way to represent sequences is to use a fixed cue for each position of the sequence. To store $\tilde{\mathbf{a}}\tilde{\mathbf{b}}\tilde{\mathbf{c}}$, the trace is:

$$\tilde{\mathbf{p}}_1 \circledast \tilde{\mathbf{a}} + \tilde{\mathbf{p}}_2 \circledast \tilde{\mathbf{b}} + \tilde{\mathbf{p}}_3 \circledast \tilde{\mathbf{c}}.$$

The retrieval (and storage) cues $\tilde{\mathbf{p}}_i$ can be arbitrary or generated in some manner from a single vector, e.g., $\tilde{\mathbf{p}}_i = (\tilde{\mathbf{p}})^i$.[11]

These methods for representing sequences can also be used to represent stacks. For example, a stack of n items, $\tilde{\mathbf{x}}_1 \cdots \tilde{\mathbf{x}}_n$, with $\tilde{\mathbf{x}}_1$ on top, can be represented by

$$\tilde{\mathbf{s}} = \tilde{\mathbf{x}}_1 + \tilde{\mathbf{p}} \circledast \tilde{\mathbf{x}}_2 + \tilde{\mathbf{p}} \circledast \tilde{\mathbf{p}} \circledast \tilde{\mathbf{x}}_3 + \cdots + \tilde{\mathbf{p}}^n \circledast \tilde{\mathbf{x}}_n.$$

The operations for manipulating such a stack are:

$$\begin{aligned} \text{push}(\tilde{\mathbf{s}}, \tilde{\mathbf{x}}) &= \tilde{\mathbf{x}} + \tilde{\mathbf{p}} \circledast \tilde{\mathbf{s}} \\ \text{top}(\tilde{\mathbf{s}}) &= \text{clean} - \text{up}(\tilde{\mathbf{s}}) \\ \text{pop}(\tilde{\mathbf{s}}) &= (\tilde{\mathbf{s}} - \text{top}(\tilde{\mathbf{s}})) \circledast \tilde{\mathbf{p}}^* \end{aligned}$$

[10] Although most of this paper assumes items are represented as real vectors, convolution memories also work with binary vectors [37].

[11] The power of a vector is defined in Section VIII.E. When using cues of the form $\tilde{\mathbf{p}}_i = (\tilde{\mathbf{p}})^i$ care must taken since the length of $(\tilde{\mathbf{p}})^i$ can increase exponentially with i.

An empty stack is noticed when the clean operation finds nothing similar to $\tilde{\mathbf{s}}$.

A problem with this type of stack implementation is that $\text{pop}(\text{push}(\tilde{\mathbf{s}}, \tilde{\mathbf{x}})) = \tilde{\mathbf{s}} \circledast \tilde{\mathbf{p}} \circledast \tilde{\mathbf{p}}^*$ is only approximately equal to $\tilde{\mathbf{s}}$. This is because $\tilde{\mathbf{p}}^*$ is an approximate inverse. A consequence is that successive pushes and pops at one level lead to the continual degradation of the lower level items. After a pair of push-pop actions, the stack will be $\tilde{\mathbf{s}} \circledast \tilde{\mathbf{p}} \circledast \tilde{\mathbf{p}}^*$, which is only approximately equal to $\tilde{\mathbf{s}}$. Additional push-pop pairs further corrupt the remaining part of the stack. There are two possible solutions to this problem – use chunking (see next section) or restrict $\tilde{\mathbf{p}}$ to be a vector for which the exact inverse is equal to the approximate inverse, in which case $\tilde{\mathbf{s}} \circledast \tilde{\mathbf{p}} \circledast \tilde{\mathbf{p}}^* = \tilde{\mathbf{s}}$ (see Section VIII.C).

V.B Chunking of sequences

All of the above methods have soft limits on the length of sequences that can be stored. As the sequences get longer the noise in the retrieved items increases until the items are impossible to identify. This limit can be overcome by chunking — creating new "non terminal" items representing subsequences [21].

The second sequence representation method is the more suitable one to do chunking with. Suppose we want to represent the sequence $\tilde{\mathbf{a}}\tilde{\mathbf{b}}\tilde{\mathbf{c}}\tilde{\mathbf{d}}\tilde{\mathbf{e}}\tilde{\mathbf{f}}\tilde{\mathbf{g}}\tilde{\mathbf{h}}$. We can create three new items representing subsequences:

$$\begin{aligned} \tilde{\mathbf{s}}_{abc} &= \tilde{\mathbf{a}} + \tilde{\mathbf{a}} \circledast \tilde{\mathbf{b}} + \tilde{\mathbf{a}} \circledast \tilde{\mathbf{b}} \circledast \tilde{\mathbf{c}} \\ \tilde{\mathbf{s}}_{de} &= \tilde{\mathbf{d}} + \tilde{\mathbf{d}} \circledast \tilde{\mathbf{e}} \\ \tilde{\mathbf{s}}_{fgh} &= \tilde{\mathbf{f}} + \tilde{\mathbf{f}} \circledast \tilde{\mathbf{g}} + \tilde{\mathbf{f}} \circledast \tilde{\mathbf{g}} \circledast \tilde{\mathbf{h}} \end{aligned}$$

These new items must be added to the item memory. The representation for the whole sequence is:

$$\tilde{\mathbf{s}}_{abc} + \tilde{\mathbf{s}}_{abc} \circledast \tilde{\mathbf{s}}_{de} + \tilde{\mathbf{s}}_{abc} \circledast \tilde{\mathbf{s}}_{de} \circledast \tilde{\mathbf{s}}_{fgh}.$$

Decoding this chunked sequence is slightly more difficult, requiring the use of a stack and decisions on whether an item is a non-terminal that should be further decoded. A machine to decode such representations is described in Section VII.B.

V.C Variable binding

It is simple to implement variable binding with convolution: convolve the variable representation with the value representation. For example, the binding of the value $\tilde{\mathbf{a}}$ to the variable $\tilde{\mathbf{x}}$ and the value $\tilde{\mathbf{b}}$ to the variable $\tilde{\mathbf{y}}$ is

$$\tilde{\mathbf{t}} = \tilde{\mathbf{x}} \circledast \tilde{\mathbf{a}} + \tilde{\mathbf{y}} \circledast \tilde{\mathbf{b}}.$$

Variables can be unbound by convolving the binding with the approximate inverse of the variable. This binding method allows multiple instances of variable in trace to be substituted for in a single-operation (approximately).

Non-recursive variable binding can also be implemented easily in other types of associative memory, e.g., the triple-space of BoltzCONS [35], or the outer product of roles and fillers in DUCS [34].

V.D Simple frame (slot/filler) structures

Simple frame-like structures can be represented using convolution encoding in a manner analogous to cross products of roles and fillers in Hinton [11] or the frames of DUCS [34]. A frame consists of a frame label and a set of roles, each represented by a vector. An instantiated frame is the sum of the frame label and the roles (slots) convolved with their respective fillers. For example, suppose we have a (very simplified) frame for "eating". The vector for the frame label is **eat** and the vectors for the roles are $\mathbf{agt}_{eat}$ and $\mathbf{obj}_{eat}$. This frame can be instantiated with the fillers **mark** and **the_fish**, to represent "Mark ate the fish":

$$\tilde{\mathbf{s}}_1 = \mathbf{eat} + \mathbf{agt}_{eat} \circledast \mathbf{mark} + \mathbf{obj}_{eat} \circledast \mathbf{the\_fish}$$

Fillers (or roles) can be retrieved from the instantiated frame by convolving with the approximate inverse of the role (or filler). The role vectors for different frames can be frame specific, i.e., $\mathbf{agt}_{eat}$ can be different from $\mathbf{agt}_{see}$, or they can be the same (or just similar).

A role filler binding such as $\mathbf{agt}_{eat} \circledast \mathbf{mark}$ is uncorrelated with either the role or the filler, because the expected value of $\tilde{\mathbf{x}} \circledast \tilde{\mathbf{y}} \cdot \tilde{\mathbf{x}}$ is zero. If it is desired that the representation for a frame be somewhat similiar to its fillers they can be added in an appropriate proportion.

V.E Recursive Frames: Holographic Reduced Representations

The vector representation of a frame is of the same dimension as the vector representation of a filler and can be used as a filler in another frame. In this way, convolution encoding affords the representation of hierarchical structure in a fixed width vector.[12]

For example, we can use an instantiated frame[13] from the previous section as a filler in another frame representing "Hunger caused Mark to eat the fish.":

$$\begin{aligned} \tilde{\mathbf{s}}_2 &= \mathbf{cause} + \mathbf{agt}_{cause} \circledast \mathbf{hunger} + \mathbf{obj}_{cause} \circledast \tilde{\mathbf{s}}_1 \\ &= \mathbf{cause} + \mathbf{agt}_{cause} \circledast \mathbf{hunger} + \mathbf{obj}_{cause} \circledast \mathbf{eat} \\ &\quad + \mathbf{obj}_{cause} \circledast \mathbf{agt}_{eat} \circledast \mathbf{mark} \\ &\quad + \mathbf{obj}_{cause} \circledast \mathbf{obj}_{eat} \circledast \mathbf{the\_fish} \end{aligned}$$

The decoding of this and other frames is discussed in Section X, where simulation results are also given.

These recursive representations can be manipulated with or without chunking. Without chunking, we could extract the agent of the object by convolving with $(\mathbf{obj}_{cause} \circledast \mathbf{agt}_{eat})^* = \mathbf{obj}_{cause}{}^* \circledast \mathbf{agt}_{eat}{}^*$. Using chunking, we could first extract the object, clean it up, and then extract its agent, giving a less noisy result. There is a tradeoff between accuracy and speed — if intermediate chunks are not cleaned up the retrievals are faster but less accurate.

[12] Slack [31] suggests a distributed memory representation for trees involving convolution products that is similar to the representation suggested here, except that it uses non-circular convolution, and thus does not work with fixed width vectors.

[13] Normalization of Euclidean lengths of the frame becomes an issue, see Section X.E.

The commutativity of the circular convolution operation can cause ambiguity in some situations. This results from the fact that $\tilde{\mathbf{t}} \circledast \tilde{\mathbf{r}}_1^* \circledast \tilde{\mathbf{r}}_2^* = \tilde{\mathbf{t}} \circledast \tilde{\mathbf{r}}_2^* \circledast \tilde{\mathbf{r}}_1^*$. The ambiguity is greatly alleviated by using frame specific role vectors rather than generic role vectors (e.g., a generic "agent" vector.) A situation when ambiguity can still arise is when two instantiations of the same frame are nested in another instantiation of that same frame. In this case the agent of the object can be confused with the object of the agent. Whether this causes problems remains to be seen. In any case, there are variants of circular convolution that are not commutative (Section VIII.G).

Holographic reduced representations provide a way of realizing of Hinton's [12] hypothetical system that could, in the same physical set of units, either focus attention on constituents or have the whole meaning present at once. Furthermore, the systematic relationship between the representations for components and frames (i.e., reduced descriptions) means that frames do not need to be decoded to gain some information about the components (see Section X.B).

VI Constraints on the vectors and the representation of features, types, and tokens

In many connectionist systems, the vectors representing items are analyzed in terms of "micro-features". For example, Hinton's family trees network [12] learned micro-features representing concepts such as age and nationality. The requirement of HRRs that elements of vectors be randomly and independently distributed seems at odds with this interpretation. Furthermore, if every element of a vector is regarded as a "micro-feature" it is unclear how to use the large number of them that the vectors of HRRs provide. This section describes a way of resolving these issues.

VI.A The representation of features, types, and tokens

There is no requirement that single micro-features be represented by single bits in a distributed representation. Features also can be represented by high-dimensional distributed representations as wide as the representation of the whole object. An item having some features can be partly the sum of those features. Tokens of a type can be distinguished from each by the addition of some identity-giving vector that is unique for each token. Features can be represented by random vectors. For example, the person "Mark" can be represented by $\mathbf{mark} = \mathbf{being} + \mathbf{person} + \mathbf{id}_{mark}$, where $\mathbf{id}_{mark}$ is some random vector that distinguishes **mark** from representations of other people. Each component feature can be weighted according to its importance or salience, if necessary.

Advantages of this scheme over a local micro-feature representation are:

- The representation of any feature of an item will degrade gracefully as the elements of the vector representing the item are corrupted.
- The number of features in an item is only loosely related to the dimensionality of the vectors representing items.
- The vectors can be of as high a dimension as desired, and higher dimensionality will give better fidelity in the representation of features.
- The vectors representing items can be expressed as sums of vectors with random independently distributed elements.

When a set of vectors representing items is constructed from distributed features in this way the elements of the vectors will not be consistent with being drawn from independent distributions. However, if linear circular convolution is used to construct representations, all the expressions describing the recall and matching of vectors can be expanded to be in terms of the random feature vectors. Thus the means and variances for the signals in a system with non-random vectors, and consequently the probabilities of correct retrieval, can be analytically derived. This is done for an example in Section X.D.

This idea of distributing features over the entire vector representing an item is not new. It is a linear transform and has been suggested by other authors under the name "randomization" or "random maps" (e.g., Schönemann [30]).

Care must be taken that the "ownership" of features is not confused when using this method to represent features (or attributes) of objects. Ambiguity of feature ownership can arise when multiple objects are stored in an addition memory. For example, suppose color and shape are encoded as additive components. If the representations for "red circle" and "blue triangle" were summed, the result would be the same as for the sum of "red triangle" and "blue circle". However, note that if the representations were convolved with distinct vectors (e.g., different role vectors) before they were added the results would not be ambiguous.

VI.B Constraints on vectors

Some authors have argued that the constraints on vectors necessary for holographic memories to perform well are too restrictive for holographic memories to be useful, e.g., [8]. This argument is based on the entirely valid observation that most vectors produced by sensory apparatus are unlikely to satisfy these constraints.

However, this argument is made in the context of storing associations between pairs of items, and is not entirely applicable to the task of storing the types of complex and structured associations that HRRs are designed for. Matrix memories have the problem of expanding dimensionality when used for this latter task, and thus do not provide a clearly superior alternative as they do in the case of storing pairwise associations.

If it is desired to interface a system which uses HRRs with another system that uses vector representations which do not conform to the constraints (e.g., a perceptual system), a hetero-associative memory can be used to translate between representations. The combination of a holographic memory (for HRRs) and a matrix based hetero-associative memory (for mapping between non-conforming and conforming representations) allows the

representation of complex associations that are difficult to represent with matrix memories alone.

VII Simple Machines that use HRRs

In this section two simple machines that operate on HRRs are described. Both of these machines have been successfully simulated on a convolution calculator using vectors with 1024 elements. The control sequencing of the second machine was done manually.

It is important to understand that HRRs are a representation for cohesive chunks. For example, HRRs can be used as a representation for the graphemic structure of words, but they are not a suitable representation for a long unstructured list of words. A long list or large set is best stored in some other type of associative memory.

VII.A Role/filler selector

To manipulate frames with roles and fillers one must be able to select the appropriate roles and fillers before convolving them. I describe here a machine which can extract the most appropriate role from an uninstantiated frame for a particular filler. The most appropriate role might be either the "first" role in the frame, or the role that combines best with the given filler. Both of these selection criteria can be combined in a single mechanism. An uninstantiated frame is stored as the sum of the roles and a frame label. Each role and filler also must be stored separately in item memory. This machine demonstrates one way in which high-level choices can be made in parallel in systems using HRRs.

Let the uninstantiated frame be $\beta\tilde{\mathbf{l}}+\alpha_1\tilde{\mathbf{r}}_1+\alpha_2\tilde{\mathbf{r}}_2+\alpha_3\tilde{\mathbf{r}}_3$, where $\tilde{\mathbf{l}}$ is the frame label, the $\tilde{\mathbf{r}}_i$ are role vectors, and the α_i and β are scalar constants. The task is to select the role that combines best with $\tilde{\mathbf{f}}$, the filler. Suppose there is some item $\tilde{\mathbf{r}}_2 \circledast \tilde{\mathbf{f}}'$ in an item memory containing roles and typical role bindings, where $\tilde{\mathbf{f}}'$ is quite similar to $\tilde{\mathbf{f}}$. The presence of a similar binding in the item memory defines $\tilde{\mathbf{r}}_2$ as the "best fitting" role for $\tilde{\mathbf{f}}$.

If the roles in the frame should be selected according to best fit, then the α_i should be approximately equal, but if $\tilde{\mathbf{r}}_1$ should be selected first, then α_1 should be greater.

The selection of the role is done by convolving the uninstantiated frame with the potential filler, i.e., $\tilde{\mathbf{f}} \circledast (\beta\tilde{\mathbf{l}} + \alpha_1\tilde{\mathbf{r}}_1 + \alpha_2\tilde{\mathbf{r}}_2 + \alpha_3\tilde{\mathbf{r}}_3)$. The closest matching vector in the item memory is $\tilde{\mathbf{r}}_2 \circledast \tilde{\mathbf{f}}'$. This can be convolved with with $\tilde{\mathbf{f}}^*$ to give a vector which can be written as $\gamma\tilde{\mathbf{r}}_2 + \tilde{\eta}$ where γ and the magnitude of the noise $\tilde{\eta}$ depend on the similarity of $\tilde{\mathbf{f}}$ to $\tilde{\mathbf{f}}'$.

This result $(\gamma\tilde{\mathbf{r}}_2 + \tilde{\eta})$ is added to the uninstantiated frame to give $\beta\tilde{\mathbf{l}}+\alpha_1\tilde{\mathbf{r}}_1+(\alpha_2+\gamma)\tilde{\mathbf{r}}_2+\alpha_3\tilde{\mathbf{r}}_3+\tilde{\mathbf{v}}_{noise}$. The strongest role can be selected by cleaning up in the item memory. Which role is represented most strongly in this trace will depend on which of α_1, $(\alpha_2 + \gamma)$, and α_3 is greater. If the α_i were approximately equal and $\tilde{\mathbf{f}}'$ was quite similar to $\tilde{\mathbf{f}}$ then $\tilde{\mathbf{r}}_2$ will be the strongest. If one of the α_i was larger or if $\tilde{\mathbf{f}}'$ was not very similar to $\tilde{\mathbf{f}}$ then the role with the largest α_i will be selected.

The machine that accomplishes this operation is shown in Figure 7.

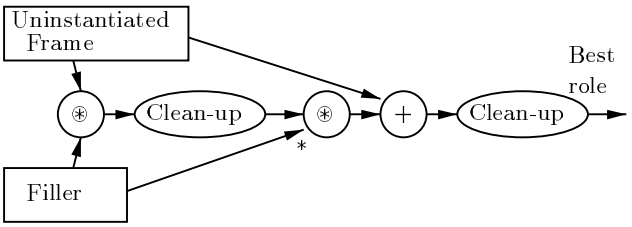


Figure 7: A role selection machine. The "*" on the operand to the convolution indicates the approximate inverse is taken.

It is possible to modify this technique to do approximate Bayesian reasoning, where the roles and fillers correspond to the hypothesis and evidence, respectively. This requires a more powerful clean-up memory that can output a blend of items, with the strength of each item in the blend proportional to the product of its strength in the input and a value associated with the item in the memory. The items in the blending clean-up memory are $\mathbf{E} \circledast \mathbf{H}_i$ (evidence convolved with hypothesis i), and the value associated with each is $\Pr(\mathbf{E} \mid \mathbf{H}_i)$. To evaluate the likelihoods of a set of hypotheses $\mathbf{H}_1, ..., \mathbf{H}_k$ given some evidence $\mathbf{E}$, one convolves the evidence with the sum of the hypothesis weighted by their prior probabilities, and passes this vector through the blending clean-up memory. After convolving this result with the approximate inverse of $\mathbf{E}$, one will have the sum of hypothesis weighted by their (approximate) relative likelihoods. This scheme suffers from the drawback that it is not possible to reverse the roles of evidence and hypothesis (and thus compute the most appropriate filler for a given role). This is because circular convolution is commutative, which means that $\Pr(\mathbf{E} \mid \mathbf{H})$ cannot be distinguished from $\Pr(\mathbf{H} \mid \mathbf{E})$ in the blending clean-up memory. This drawback could be overcome by using one of the non-commutative variants of circular convolution (Section VIII.G).

VII.B Chunked sequence readout machine

A machine that reads out the chunked sequences described in Section V.B can be built using two buffers, a stack, a classifier, a correlator, a clean up memory, and three gating paths. The classifier tells whether the item most prominent in the trace is a terminal, a non-terminal (chunk) or nothing. At each iteration the machine executes one of three action sequences depending on the output of the classifier. The stack could be implemented in any of a number of ways; including the way suggested earlier or in a simple addition memory. The machine is shown in Figure 8.

The control loop for the chunked sequence readout machine is:

Loop: (until stack gives END signal)

> Clean up the trace to recover most prominent item: $\tilde{\mathbf{x}} = \text{Clean}(\tilde{\mathbf{t}})$.
>
> Classify $\tilde{\mathbf{x}}$ as a terminal, non-terminal, or "nothing" (in which case "pop" is the appropriate action) and do the appropriate of the following action sequences.

Terminal:

1 Item $\tilde{\mathbf{x}}$ is on output. T1 gates path to replace trace by its follower: $\tilde{\mathbf{t}} \leftarrow \tilde{\mathbf{x}}^* \circledast (\tilde{\mathbf{t}} - \tilde{\mathbf{x}})$.

Non-terminal:

1 Signal N1 tells stack to push the follower of the non terminal: $\tilde{\mathbf{s}} \leftarrow \text{push}(\tilde{\mathbf{s}}, \tilde{\mathbf{x}}^* \circledast (\tilde{\mathbf{t}} - \tilde{\mathbf{x}}))$.

2 Signal N2 gates path to replace trace by the non-terminal: $\tilde{\mathbf{t}} \leftarrow \tilde{\mathbf{x}}$.

Pop:

1 Signal P1 gates path to replace trace by top of stack: $\tilde{\mathbf{t}} \leftarrow \text{top}(\tilde{\mathbf{s}})$.

2 Signal P2 tells stack to discard top of stack: $\tilde{\mathbf{s}} \leftarrow \text{pop}(\tilde{\mathbf{s}})$. Stack gives END signal if empty.

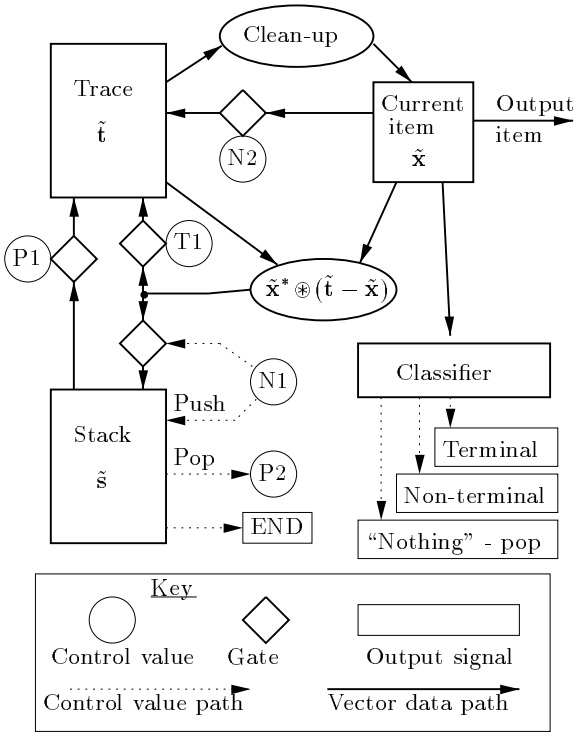


Figure 8: A chunked sequence readout machine. A simple controller (not shown but described in text) receives classifier output and provides boolean control values P1, P2, T1, N1, and N2.

In this machine the trace buffer contains the chunk currently being decoded, and the stack contains the portions of higher level chunks that are yet to be decoded.

The chunked sequence readout machine is an example of a system that achieves Hinton's [12]

objectives of being able to focus attention on constituents when necessary or have the whole "meaning" of a chunk present at once.

VIII Mathematical Properties

Circular convolution may be regarded as a multiplication operation over vectors: two vectors multiplied together (convolved) result in another vector. A finite dimensional vector space over the real numbers, with circular convolution as multiplication and the usual definitions of scalar multiplication and vector addition, forms a commutative linear algebra. This is most easily proved using the observation that convolution corresponds to element-wise multiplication in a different basis, as described in Section VIII.B. All the rules that apply to scalar algebra (i.e., associativity and commutativity of addition and multiplication, and distributivity of multiplication over addition) also apply to this algebra. This makes it very easy to manipulate expressions containing additions, convolutions, and scalar multiplications.

This algebra has many of the same properties as the algebra considered by Borsellino and Poggio [3] and Shönemann [30], which had aperiodic convolution as a multiplication operation over an infinite dimensional vector space restricted to vectors with a finite number of non-zero elements. Shönemann observed that representing the correlation of $\tilde{\mathbf{b}}$ and $\tilde{\mathbf{a}}$ as a convolution of $\tilde{\mathbf{a}}$ with an involution of $\tilde{\mathbf{b}}$ made expressions with convolutions and correlations easier to manipulate.

VIII.A Distributions of dot products

The distributions of the dot products of vectors and convolutions of vectors can be analytically derived. Some useful dot products are shown in Table 1. The variances and means shown are based on the assumption that the elements for the vectors $\tilde{\mathbf{a}}, \tilde{\mathbf{b}}, \tilde{\mathbf{c}}$, and $\tilde{\mathbf{d}}$ are independently distributed as $\text{N}(0, \frac{1}{n})$. It follows that the expected length of these vectors is 1. Dot products are the sum of scalar products of individual vector elements, and are thus normally distributed, by the central limit theorem.[14] The variance of a dot product term depends upon the number of correlated scalar products in the dot product. The equivalent expressions in rows (5) to (10) are derived from the following identity of convolution algebra:

$$\tilde{\mathbf{a}} \circledast \tilde{\mathbf{b}} \circledast \tilde{\mathbf{x}}^* \cdot \tilde{\mathbf{y}} \;=\; \tilde{\mathbf{a}} \circledast \tilde{\mathbf{b}} \circledast \tilde{\mathbf{y}}^* \cdot \tilde{\mathbf{x}} \;=\; \tilde{\mathbf{a}} \circledast \tilde{\mathbf{b}} \cdot \tilde{\mathbf{x}} \circledast \tilde{\mathbf{y}}$$

These means and variances are used as follows: Suppose that $\tilde{\mathbf{a}}, \tilde{\mathbf{b}}, \tilde{\mathbf{c}}, \tilde{\mathbf{d}}$, and $\tilde{\mathbf{e}}$ are random vectors with elements drawn independently from $\text{N}(0, \frac{1}{n})$. Then the value of $\tilde{\mathbf{a}}^* \circledast (\tilde{\mathbf{a}} \circledast \tilde{\mathbf{b}} + \tilde{\mathbf{c}} \circledast \tilde{\mathbf{d}}) \cdot \tilde{\mathbf{b}}$ will have an expected value of 1 and a variance of $\frac{7n+4}{n^2}$ $(= \frac{6n+4}{n^2} + \frac{1}{n}$ using rows 6 and 10 in Table 1). The value of $\tilde{\mathbf{a}}^* \circledast (\tilde{\mathbf{a}} \circledast \tilde{\mathbf{b}} + \tilde{\mathbf{c}} \circledast \tilde{\mathbf{d}}) \cdot \tilde{\mathbf{e}}$ will have an expected value of 0 and a variance of $\frac{3n+2}{n^2}$ $(= \frac{2n+2}{n^2} + \frac{1}{n}$ using rows 8 and 10 in Table 1).

Of some interest is the distribution of the elements of $\tilde{\mathbf{a}} \circledast \tilde{\mathbf{b}}$. If the elements of $\tilde{\mathbf{a}}$ and $\tilde{\mathbf{b}}$ are independently distributed as $\text{N}(0, \frac{1}{n})$ then the mean of the elements of $\tilde{\mathbf{a}} \circledast \tilde{\mathbf{b}}$ is zero but the variance is higher than $1/n$ and the covariance of the elements is not zero. The expected length of $\tilde{\mathbf{a}} \circledast \tilde{\mathbf{b}}$ is still one, provided that the elements of $\tilde{\mathbf{a}}$ are distributed independently of those of $\tilde{\mathbf{b}}$ (the expected length of $\tilde{\mathbf{a}} \circledast \tilde{\mathbf{a}}$ is $\sqrt{(2n+2)/n}$). Thus, the variance of $\tilde{\mathbf{a}} \circledast \tilde{\mathbf{b}} \circledast \tilde{\mathbf{c}} \circledast \tilde{\mathbf{a}}^* \circledast \tilde{\mathbf{b}}^* \cdot \tilde{\mathbf{c}}$ is higher than that of $\tilde{\mathbf{a}} \circledast \tilde{\mathbf{b}} \circledast \tilde{\mathbf{a}}^* \cdot \tilde{\mathbf{b}}$. A consequence of this is that some care must be taken

[14] Although there are correlations among these scalar products there is sufficient independence for the central limit theorem to apply.

	Expression	mean	variance
(1)	$\tilde{\mathbf{a}} \cdot \tilde{\mathbf{a}}$	1	$\frac{2}{n}$
(2)	$\tilde{\mathbf{a}} \cdot \tilde{\mathbf{b}}$	0	$\frac{1}{n}$
(3)	$\tilde{\mathbf{a}} \cdot \tilde{\mathbf{a}} \circledast \tilde{\mathbf{b}}$	0	$\frac{2n+1}{n^2}$
(4)	$\tilde{\mathbf{a}} \cdot \tilde{\mathbf{b}} \circledast \tilde{\mathbf{c}}$	0	$\frac{1}{n}$
(5)	$\tilde{\mathbf{a}} \circledast \tilde{\mathbf{a}} \cdot \tilde{\mathbf{a}} \circledast \tilde{\mathbf{a}} = \tilde{\mathbf{a}} \circledast \tilde{\mathbf{a}} \circledast \tilde{\mathbf{a}}^* \cdot \tilde{\mathbf{a}}$	$\frac{2n+2}{n}$	$\frac{40n+112}{n^2}$
(6)	$\tilde{\mathbf{a}} \circledast \tilde{\mathbf{b}} \cdot \tilde{\mathbf{a}} \circledast \tilde{\mathbf{b}} = \tilde{\mathbf{a}} \circledast \tilde{\mathbf{b}} \circledast \tilde{\mathbf{a}}^* \cdot \tilde{\mathbf{b}}$	1	$\frac{6n+4}{n^2}$
(7)	$\tilde{\mathbf{a}} \circledast \tilde{\mathbf{b}} \cdot \tilde{\mathbf{a}} \circledast \tilde{\mathbf{a}} = \tilde{\mathbf{a}} \circledast \tilde{\mathbf{b}} \circledast \tilde{\mathbf{a}}^* \cdot \tilde{\mathbf{a}}$	0	$\frac{6n+18}{n^2}$
(8)	$\tilde{\mathbf{a}} \circledast \tilde{\mathbf{b}} \cdot \tilde{\mathbf{a}} \circledast \tilde{\mathbf{c}} = \tilde{\mathbf{a}} \circledast \tilde{\mathbf{b}} \circledast \tilde{\mathbf{a}}^* \cdot \tilde{\mathbf{c}}$	0	$\frac{2n+2}{n^2}$
(9)	$\tilde{\mathbf{a}} \circledast \tilde{\mathbf{b}} \cdot \tilde{\mathbf{c}} \circledast \tilde{\mathbf{c}} = \tilde{\mathbf{a}} \circledast \tilde{\mathbf{b}} \circledast \tilde{\mathbf{c}}^* \cdot \tilde{\mathbf{c}}$	0	$\frac{2n+2}{n^2}$
(10)	$\tilde{\mathbf{a}} \circledast \tilde{\mathbf{b}} \cdot \tilde{\mathbf{c}} \circledast \tilde{\mathbf{d}} = \tilde{\mathbf{a}} \circledast \tilde{\mathbf{b}} \circledast \tilde{\mathbf{c}}^* \cdot \tilde{\mathbf{d}}$	0	$\frac{1}{n}$

Table 1: Means and variances of dot products of common convolution expressions. All are normally distributed. n is the dimensionality of the vectors.

when using $\tilde{\mathbf{a}} \circledast \tilde{\mathbf{b}}$ as a storage cue, especially in the case where $\tilde{\mathbf{a}} = \tilde{\mathbf{b}}$. This is particularly relevant to the storage capabilities of HRRs because when recursive frames are stored, convolution products e.g., $\mathbf{obj}_{cause} \circledast \mathbf{agt}_{eat}$, are the storage cues.

VIII.B Using FFTs to compute convolution

The fastest way to compute convolution is via Fast Fourier transforms (FFT) [4]. The computation involves a transform, an element-wise multiplication of two vectors, and an inverse transform. We can write

$$\tilde{\mathbf{a}} \circledast \tilde{\mathbf{b}} = \tilde{\mathbf{f}}'(\tilde{\mathbf{f}}(\tilde{\mathbf{a}}) \odot \tilde{\mathbf{f}}(\tilde{\mathbf{b}})),$$

where $\tilde{\mathbf{f}}$ is a discrete Fourier transform, $\tilde{\mathbf{f}}'$ is the inverse discrete Fourier transform, and $\odot$ is the element-wise multiplication of two vectors.

These three steps take $O(n \log n)$ time to compute, whereas the obvious implementation of the convolution equation $c_i = \sum_j a_j b_{i-j}$ takes $O(n^2)$ time to compute.[15]

I shall refer to the original domain as the *spatial domain*, and the domain the Fourier transform takes it to as the *frequency domain*. Both domains are n-dimensional vector spaces, and both the forward and inverse Fourier transforms are linear.

The discrete Fourier transform, $\tilde{\mathbf{f}} : \mathcal{C}^n \rightarrow \mathcal{C}^n$, ($\mathcal{C}$ is the field of complex numbers) is defined as:

$$f_j(\tilde{\mathbf{x}}) = \sum_{k=0}^{n-1} x_k e^{-i2\pi jk/n}$$

where $i^2 = -1$ and $f_j(\tilde{\mathbf{x}})$ is the jth element of $\tilde{\mathbf{f}}(\tilde{\mathbf{x}})$. The discrete Fourier transform is invertible and defines a one to one relationship between vectors in the spatial and frequency domains. It can be computed in $O(n \log n)$ time using the Fast Fourier Transform (FFT) algorithm.

[15] Computing convolution via FFTs takes about the same time as the $O(n^2)$ method for $n = 32$. It is faster for $n > 32$ and slower for $n < 32$.

The inverse discrete Fourier transform is very similar:

$$f_j^{-1}(\tilde{\mathbf{x}}) = \frac{1}{n} \sum_{k=0}^{n-1} x_k e^{i2\pi jk/n}$$

and can also be computed in $O(n \log n)$ time using the FFT algorithm.

VIII.C Identities and approximate and exact inverses in the frequency domain

Since convolution in the spatial domain is equivalent to element-wise multiplication in the frequency domain we can easily find convolutive inverses by transforming into the frequency domain. By definition $\tilde{\mathbf{y}}$ is the inverse of $\tilde{\mathbf{x}}$ if $\tilde{\mathbf{x}} \circledast \tilde{\mathbf{y}} = \tilde{\mathbf{1}}$ and we can write $\tilde{\mathbf{y}} = \tilde{\mathbf{x}}^{-1}$. The convolutive identity vector is $\tilde{\mathbf{1}} = (1, 0, ..., 0)$. Transforming this into the frequency domain gives

$$\tilde{\mathbf{f}}(\tilde{\mathbf{x}}) \odot \tilde{\mathbf{f}}(\tilde{\mathbf{x}}^{-1}) = \tilde{\mathbf{f}}(\tilde{\mathbf{1}}).$$

The transform of the identity is

$$\tilde{\mathbf{f}}(\tilde{\mathbf{1}}) = (e^{0i}, e^{0i}, ..., e^{0i}) = (1, 1, ..., 1).$$

This gives independent relationships between the corresponding elements of $\tilde{\mathbf{f}}(\tilde{\mathbf{x}})$ and $\tilde{\mathbf{f}}(\tilde{\mathbf{x}}^{-1})$ which can be expressed as

$$f_j(\tilde{\mathbf{x}}^{-1}) f_j(\tilde{\mathbf{x}}) = 1.$$

Expressing $\tilde{\mathbf{f}}(\tilde{\mathbf{x}})$ in polar coordinates gives

$$f_j(\tilde{\mathbf{x}}) = r_j e^{i\theta_j}$$

and we can see that the Fourier transform of the inverse of $\tilde{\mathbf{x}}$ is

$$f_j(\tilde{\mathbf{x}}^{-1}) = \frac{1}{r_j} e^{-i\theta_j}.$$

Now consider the approximate inverse. It can be seen from the definition of the Fourier transform that the transform of the involution of $\tilde{\mathbf{x}}$ is

$$f_j(\tilde{\mathbf{x}}^*) = r_j e^{-i\theta_j}.$$

The difference in the frequency domain between the approximate inverse and the exact inverse is that the elements of the approximate inverse have the same magnitudes as the original elements, whereas the magnitudes of the elements of the exact inverse are the reciprocals of the magnitudes of the original elements. It follows that the involution gives the exact inverse when $r_j = 1$, i.e., when $\mid f_j(\tilde{\mathbf{x}}) \mid = 1$. I refer to this class of vectors as *unitary* vectors.[16] An equivalent condition is that the auto correlation of $\tilde{\mathbf{x}}$ is the convolutive identity vector (i.e., the delta function with magnitude 1).

VIII.D Why the exact inverse is not always useful

Since $\tilde{\mathbf{a}}^{-1}$ can be used to decode $\tilde{\mathbf{a}} \circledast \tilde{\mathbf{b}}$ exactly, it might seem to be a better candidate for the decoding vector than the approximate inverse $\tilde{\mathbf{a}}^*$. However, using the exact inverse results in a lower signal-to-noise ratio in

[16] In analogy with unitary matrices, whose Hermitian conjugates (complex conjugate of transpose) equal their inverses.

the retrieved vector when the memory trace is noisy or when there are other vectors added into it. This problem arises because, for vectors with elements independently distributed as $\mathrm{N}(0, \frac{1}{n})$, $|\tilde{\mathbf{a}}^*|$ always equals $|\tilde{\mathbf{a}}|$, but $|\tilde{\mathbf{a}}^{-1}|$ is usually greater than $|\tilde{\mathbf{a}}|$, except for unitary vectors. This is not unexpected, as inverse filter are well known to be sensitive to noise [7].

VIII.E The convolutive power of a vector

The convolutive power of a vector (exponentiation) is straightforwardly defined by exponentiation of its elements in the frequency domain, i.e.,

$$f_j(\tilde{\mathbf{x}}^k) = (f_j(\tilde{\mathbf{x}}))^k.$$

Fractional and negative exponents of vectors are defined in the same way as for complex numbers. Integer powers are useful for generating some types of encoding keys (cf. Section V.A) and fractional powers can be used to represent trajectories through continuous space [25].

VIII.F Matrices corresponding to circular convolution

The convolution operation can be expressed as a matrix-vector multiplication.

$$\tilde{\mathbf{a}} \circledast \tilde{\mathbf{b}} = M_a \tilde{\mathbf{b}}$$

where M_a is the matrix corresponding to convolution by $\tilde{\mathbf{a}}$. It has elements $m_{a_{ij}} = a_{i-j}$ (where the subscripts on $\tilde{\mathbf{a}}$ are interpreted modulo n). Such matrices are known as "circulant matrices" [6]. The eigenvalues of M_a are the individual (complex valued) elements of the Fourier transform of $\tilde{\mathbf{a}}$. The corresponding eigenvectors are the inverse transforms of the frequency components (i.e., $(1,0,0,...)$, $(0,1,0,...)$, etc. in the frequency domain). Thus it is possible for the mapping computed by the connections between two layers in a feed-forward network (i.e. a matrix multiplication) to correspond to convolution by a fixed vector.

VIII.G Non-commutative variants and analogs of convolution

The commutativity of convolution can cause ambiguity in the representations of some structures. If this is a problem, non-commutative variants of circular convolution can be computed by permuting the elements of the argument vectors in either the spatial or frequency domain. The permutations applied to right and left vectors must be different. The resulting operation is neither commutative nor associative, but is bilinear (and thus distributes over addition and preserves similarity) and has an easily computed approximate inverse.

An alternative operation that is non-commutative but still associative is matrix multiplication. This could be used to associate two vectors by treating each vector as a square matrix. The dimension of the vectors would have to be a perfect square. I am unaware of what the scaling and interference properties of such an associative memory operation would be. It would be similarity-preserving and vectors corresponding to orthogonal matrices would have simple inverses.

VIII.H Partial derivatives of convolutions

A convolution operation can be used in a feed-forward networks[17] and values can be forward-propagated in $O(n \log n)$ time (on serial machines). Derivatives can also be back-propagated in $O(n \log n)$ time. Suppose we have a network in which the values from two groups of units are convolved together and sent to a third group of units. The relevant portion of such a feed-forward network is shown in Figure 9. Suppose we have the partial derivatives $\frac{\partial E}{\partial c_i}$ of outputs of the convolution with respect to an objective function E. Then the partial derivatives of the inputs to the convolution can be calculated as follows:

$$\begin{aligned}\frac{\partial E}{\partial a_k} &= \textstyle\sum_i \frac{\partial E}{\partial c_i}\frac{\partial c_i}{\partial a_k} \\ &= \textstyle\sum_i \frac{\partial E}{\partial c_i} b_{i-k} \\ &= \textstyle\sum_i \frac{\partial E}{\partial c_i} [\tilde{\mathbf{b}}^*]_{k-i} \\ &= \left[\tilde{\partial}_c \circledast \tilde{\mathbf{b}}^*\right]_k\end{aligned}$$

where $\tilde{\partial}_c$ is the vector with elements $\frac{\partial E}{\partial c_i}$, and $[\cdot]_k$ is the kth element of a vector.

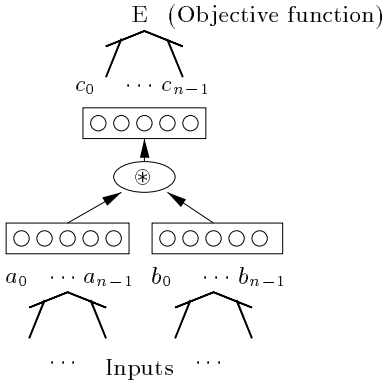


Figure 9: A convolution operation in a back-propagation network.

This means that it is possible to incorporate a convolution operation in a feed-forward network and do the forward and back-propagation computations for the convolution in $O(n \log n)$ time. One reason one might want to do this could be to use a back-propagation network to learn good vector representations for items for some specific task. This is pursued in Plate [25].

IX The capacity of convolution memories and HRRs

The number of associations that can be stored in a convolution memory is approximately linear in the dimensionality of the vectors. In Appendix B it is shown that the number of pairs of vectors that can be stored in a

[17] For an introduction to feed-forward networks see [29].

convolution memory trace is at least

$$k > \frac{n}{16 \ln \frac{m^2}{q}} - 2$$

where n is the vector dimension, m is the number of candidate vectors, and q is the probability of one or more errors in decoding. It is assumed that vector elements are independently distributed as $N(0, \frac{1}{n})$, and that any vector does not appear more than once in a trace. For a wide range of parameter values, numerical solutions of the capacity equation (Equation 8 in Appendix B.) are well approximated by

$$n = 4.5(k + 0.7) \ln \frac{m}{30q^4}.$$

If either of the assumptions are violated, that is if the vectors have similarity (i.e., are not independent), or if the same vector is stored in more than one pair, the convolution memory will still work, but the probability of error will increase. The effect of similarity among the vectors on the capacity is considered at greater length in Plate [24].

The size of a structure that can be stored in (and successfully retrieved from) a HRR increases almost linearly with the vector dimension, with similar constants to those above. The "size" is the number of terms in the expanded convolution expression (the sum of convolution products) for the structure. For example, the HRR in Section V.E has five terms. The probability of correctly decoding a deep structure is slightly less than that for correctly decoding a shallow structure with the same number of terms because the variance for decoding long convolution products, e.g., the variance of $(\tilde{\mathbf{a}} \circledast \tilde{\mathbf{b}} \circledast \tilde{\mathbf{c}}) \circledast (\tilde{\mathbf{b}} \circledast \tilde{\mathbf{c}})^* \cdot \tilde{\mathbf{a}}$ is slightly higher than that for decoding shorter convolution products. This drop in performance for deeper structures can be avoided by using unitary vectors for the encoding cues (cf. Section VIII.C).

X An example of encoding and decoding HRRs

An example of HRR frame construction and decoding is presented in this section. The types and tokens representing objects and concepts are constructed according to the suggestions in Section VI. Results from a simulation of the example using 512 dimensional vectors are reported.

X.A Representation and similarity of tokens

The suggestion in Section VI for token vectors (representing an instance of a type) was that they be composed of the sum of features and a distinguishing vector giving individual identity. In this example the base vectors (representing features) have elements chosen independently from $N(0, \frac{1}{512})$. The base vectors are listed in Table 3. The token and role vectors are constructed by summing the relevant feature vectors and a distinguishing random "identity" vector that is used to give a distinct identity to an instance of a type. Scale factors are included in order to make the expected length of the vectors equal to 1. These token vectors and a representative pair of role vectors are listed in Table 4.

The similarity matrix of the tokens is shown in Table 6. Tokens with more features in common have higher similarity (e.g., **mark** and **john**), and tokens with no features in common have very low similarity (e.g., **john** and **the_fish**).

$\tilde{\mathbf{s}}_1$	Mark ate the fish.
$\tilde{\mathbf{s}}_2$	Hunger caused Mark to eat the fish.
$\tilde{\mathbf{s}}_3$	John ate.
$\tilde{\mathbf{s}}_4$	John saw Mark.
$\tilde{\mathbf{s}}_5$	John saw the fish.
$\tilde{\mathbf{s}}_6$	The fish saw John.

Table 2: Sentences.

Object features		Role features	Frame labels
being	**food**	**obj**	**cause**
person	**fish**	**agt**	**eat**
state	**bread**		**see**

Table 3: Base feature vectors. Vector elements are all independently chosen from $N(0, 1/512)$.

$$\begin{aligned}
\textbf{mark} &= (\textbf{being} + \textbf{person} + \textbf{id}_{mark})/\sqrt{3} \\
\textbf{john} &= (\textbf{being} + \textbf{person} + \textbf{id}_{john})/\sqrt{3} \\
\textbf{paul} &= (\textbf{being} + \textbf{person} + \textbf{id}_{paul})/\sqrt{3} \\
\textbf{luke} &= (\textbf{being} + \textbf{person} + \textbf{id}_{luke})/\sqrt{3} \\
\textbf{the_fish} &= (\textbf{food} + \textbf{fish} + \textbf{id}_{the_fish})/\sqrt{3} \\
\textbf{the_bread} &= (\textbf{food} + \textbf{bread} + \textbf{id}_{the_bread})/\sqrt{3} \\
\textbf{hunger} &= (\textbf{state} + \textbf{id}_{hunger})/\sqrt{2} \\
\textbf{thirst} &= (\textbf{state} + \textbf{id}_{thirst})/\sqrt{2} \\
\textbf{agt}_{eat} &= (\textbf{agt} + \textbf{id}_{eat_agent})/\sqrt{2} \\
\textbf{obj}_{eat} &= (\textbf{obj} + \textbf{id}_{eat_object})/\sqrt{2}
\end{aligned}$$

Table 4: Token and role vectors constructed from base feature vectors and random identity-giving vectors. The identity vectors (e.g., $\mathbf{id}_{mark}$) are chosen in the same way as the base feature vectors. The denominators are chosen so that the expected length of a vector is 1.0. Other roles (e.g., $\mathbf{agt}_{see}$) are constructed in the analogous fashion.

X.B Representation and similarity of frames

The six sentences listed in Table 2 are represented as HRR frames. The expressions for these HRRs are listed in Table 5. Again, scale factors are included to make the expected length of the vectors equal to 1.

The similarities of the HRRs are shown in Table 7. Some similarities between instantiated frames can be detected without decoding – the HRRs for similar sentences (e.g., $\tilde{\mathbf{s}}_4$, $\tilde{\mathbf{s}}_5$, and $\tilde{\mathbf{s}}_6$) are similar (i.e., they have high

$$
\begin{aligned}
\tilde{\mathbf{s}}_1 &= (\mathbf{eat} + \mathbf{agt}_{eat} \circledast \mathbf{mark} + \mathbf{obj}_{eat} \circledast \mathbf{the\_fish})/\sqrt{3} \\
\tilde{\mathbf{s}}_2 &= (\mathbf{cause} + \mathbf{agt}_{cause} \circledast \mathbf{hunger} \\
&\qquad + \mathbf{obj}_{cause} \circledast \tilde{\mathbf{s}}_1)/\sqrt{3} \\
\tilde{\mathbf{s}}_3 &= (\mathbf{eat} + \mathbf{agt}_{eat} \circledast \mathbf{john})/\sqrt{2} \\
\tilde{\mathbf{s}}_4 &= (\mathbf{see} + \mathbf{agt}_{see} \circledast \mathbf{john} + \mathbf{obj}_{see} \circledast \mathbf{mark})/\sqrt{3} \\
\tilde{\mathbf{s}}_5 &= (\mathbf{see} + \mathbf{agt}_{see} \circledast \mathbf{john} + \mathbf{obj}_{see} \circledast \mathbf{the\_fish})/\sqrt{3} \\
\tilde{\mathbf{s}}_6 &= (\mathbf{see} + \mathbf{agt}_{see} \circledast \mathbf{the\_fish} + \mathbf{obj}_{see} \circledast \mathbf{john})/\sqrt{3}
\end{aligned}
$$

Table 5: HRR frame vectors representing the sentences in Table 2

	mark	**john**	**paul**	**luke**	**the_fish**	**the_bread**	**hunger**	**thirst**
mark	1.07							
john	0.78	1.08						
paul	0.76	0.75	1.08					
luke	0.73	0.68	0.74	1.01				
the_fish	0.01	0.00	-.02	-.03	1.16			
the_bread	0.02	0.01	0.06	0.01	0.35	0.97		
hunger	0.01	0.06	0.05	0.03	0.10	0.03	0.93	
thirst	0.01	0.11	0.04	0.06	0.07	0.02	0.48	1.04

Table 6: Similarities (dot-products) among some of the tokens. The diagonal elements are the squares of the vector lengths. Tokens sharing feature vectors (see Table 4) have higher similarity.

	$\tilde{\mathbf{s}}_1$	$\tilde{\mathbf{s}}_2$	$\tilde{\mathbf{s}}_3$	$\tilde{\mathbf{s}}_4$	$\tilde{\mathbf{s}}_5$	$\tilde{\mathbf{s}}_6$
$\tilde{\mathbf{s}}_1$	1.14					
$\tilde{\mathbf{s}}_2$	0.02	0.98				
$\tilde{\mathbf{s}}_3$	0.81	0.01	1.11			
$\tilde{\mathbf{s}}_4$	0.11	0.12	0.25	1.13		
$\tilde{\mathbf{s}}_5$	0.30	0.05	0.31	0.73	0.99	
$\tilde{\mathbf{s}}_6$	-.01	0.14	0.01	0.65	0.35	1.14

Table 7: Similarities (dot-products) among the frames.

dot-products.) Note that $\tilde{\mathbf{s}}_5$ and $\tilde{\mathbf{s}}_6$ have the same constituents, but are distinct because their structures are different. In fact, $\tilde{\mathbf{s}}_5$ has a higher dot-product with $\tilde{\mathbf{s}}_4$ than with $\tilde{\mathbf{s}}_6$, because $\tilde{\mathbf{s}}_5$ and $\tilde{\mathbf{s}}_4$ have the same filler in the same first role, which creates more similarity than having the same fillers in different roles.

X.C Extracting fillers and roles from frames

The filler of a particular role in a frame is extracted as follows: the frame is convolved with the approximate inverse of the role and the result is cleaned up by choosing the most similar vector in the item memory. The item memory contains all feature, token, role, and frame vectors (i.e., all the vectors listed in Tables 3, 4, and 5).

The extraction of various fillers and roles is shown in Table 8. For each extraction, the three vectors in item memory that are most similar to the result are shown. In all cases the most similar object is the correct one.

As shown in row (1), the expression to extract the agent of $\tilde{\mathbf{s}}_1$ is

$$x = \tilde{\mathbf{s}}_1 \circledast \mathbf{agt}_{eat}{}^{*}.$$

The three objects in item memory most similar to x (with their respective dot-products) are **mark** (0.62), **john** (0.47), and **paul** (0.41). The filler of the agent role in $\tilde{\mathbf{s}}_1$ is indeed **mark**, so the extraction has been performed correctly.

The construction of $\tilde{\mathbf{s}}_1$ and the determination of the filler of its object role, on row (1) in Table 8, is illustrated in Figure 10. In order to enable the perception of similarities among vectors in this figure, the 512-element vectors were laid out in rectangles with dimensions permuted (on all vectors simultaneously) so as to reduce the total sum of variance between neighboring elements. This was done using a simulated annealing program. The reader should not take the visual similarites of the vectors too seriously – dot-product similarity is what is important and is very difficult to judge from merely looking at figures like this.

Row (2) illustrates that the agent of $\tilde{\mathbf{s}}_1$ can also be extracted using the generic agent role (**agt**) rather than the agent role specific to the eat frame ($\mathbf{agt}_{eat}$). The results are stronger when the specific agent is used.

In $\tilde{\mathbf{s}}_2$ the object role is filled by another frame. There are two alternative methods for extracting the components of this sub-frame. The first method, which is slower, is to clean up the sub-frame in item memory (row 5) and then extract its components, as in rows (1) to (3). The second (faster) method, is to omit the clean up operation and directly convolve the result with the approximate inverses of the roles of $\tilde{\mathbf{s}}_1$. The expressions for the fast method are shown in rows (6) and (7). The first method is an example of using chunking to clean up intermediate results, and gives stronger results at the expense of introducing intermediate cleanup operations. With the intermediate cleanup omitted the chances of error are higher; in row (6) the correct vector is only very slightly stronger than an incorrect one. However, the high-scoring incorrect responses are similar to the correct response; it is clear that the sub-frame object role filler is a person.

Row (8) shows what happens when we try to extract the filler of an absent role. The frame $\tilde{\mathbf{s}}_3$ ("John ate.") has no object. As expected, $\tilde{\mathbf{s}}_3 \circledast \mathbf{obj}_{eat}{}^*$ is not significantly similar to anything. Although **food** might seem an appropriate guess, it is coincidence that **food** is the strongest response.

It is possible to determine which role a token is filling, as in rows (9) and (10). In $\tilde{\mathbf{s}}_4$, on row (9), the correct role for **john** is $\mathbf{agt}_{eat}$, but $\mathbf{obj}_{eat}$ also scores quite highly. This is because **john** is a person, and a person is also filling the object role in $\tilde{\mathbf{s}}_4$. Compare this with $\tilde{\mathbf{s}}_5$, where the object role filler (**the_fish**) is not at all similar to the agent role filler (**john**). The extracted role for **john** is not at all similar to the object role, as shown on row (10).

X.D Probabilities of correct decoding

The expectation and variances of the dot-products

$$\tilde{\mathbf{s}}_1 \circledast \mathbf{agt}_{eat}{}^* \cdot \mathbf{mark} \quad \text{and} \quad \tilde{\mathbf{s}}_1 \circledast \mathbf{agt}_{eat}{}^* \cdot who$$

are calculated in this section (where *who* is a vector for a person that is not **mark**). This allows us to calculate the probability that the agent of $\tilde{\mathbf{s}}_1$ will be extracted correctly, as in row (1) of Table 8. It must be emphasized that the behavior of any particular system (i.e., set of vectors) is deterministic. A particular frame in a particular system always will or always will not be decoded correctly. The probabilities calculated in this section are the probabilities that a randomly chosen system will behave in a particular way.

Let $d = \tilde{\mathbf{s}}_1 \circledast \mathbf{agt}_{eat}{}^*$, then

$$d \cdot \mathbf{mark} = d \cdot (\mathbf{being} + \mathbf{person} + \mathbf{id}_{mark})/\sqrt{3}, \text{ and}$$

$$d \cdot p = d \cdot (\mathbf{being} + \mathbf{person} + \mathbf{id}_p)/\sqrt{3}.$$

The vector p is used here as a generic "incorrect person" filler. The extraction is judged to have been performed correctly if $d \cdot \mathbf{mark} > d \cdot p \quad \forall \quad p \in$ {vectors in item memory}. We can limit the consideration of p to people-vectors in item memory, because it is extremely unlikely that other vectors will be more similar to d than the vectors representing people.

It is important to note that these two dot-products are correlated because they share the common term $d \cdot (\mathbf{being} + \mathbf{person})/\sqrt{3}$. To calculate the probabilities accurately it is necessary to take into account the value of this term when choosing the threshold. Let

$$\begin{aligned} X_{mark} &= d \cdot \mathbf{id}_{mark}/\sqrt{3}, \\ Y_p &= d \cdot \mathbf{id}_p/\sqrt{3}, \text{ and} \\ Z &= d \cdot (\mathbf{being} + \mathbf{person})/\sqrt{3}. \end{aligned}$$

The values X_{mark}, Y_p, and Z can be regarded as uncorrelated variables that are derived from the random vectors. They are distributed normally.

The calculation of the means and variances of X_{mark}, Y_p, and Z is presented in Appendix C. For $n = 512$ they are:

	mean	variance	std dev
X_{mark}	0.192	0.00116	0.0341
Y_p	0	0.000867	0.0294
Z	0.385	0.00246	0.0496

A lower estimate [18] for the probability P that $Z + X_{mark} > Z + Y_p$ for all p is given by

$$P' = \Pr(Z + X_{mark} > t) \cdot \Pr(Z + Y_p < t \ \forall \ p \neq \mathbf{mark}),$$

where t is a threshold chosen to maximize this probability.[19]

In this example there are three other people, so

$$P' = \Pr(X_{mark} + Z > t) \cdot \Pr(Y_p + Z < t)^3.$$

This has a maximum value of 0.996 for $t = Z + 0.0955$. Thus the probability of correctly identifying **mark** as the filler of agent role in $\tilde{\mathbf{s}}_1$ is at least 0.996. If there were 100 other people the probability would drop to 0.984.

The primary reason for calculating means and variances of signals is to estimate the vector dimension that will result in an acceptable probability of error. It is not necessary to calculate the means and variances for every possible signal value, only the one where the means which must be discriminated are small and the variances large.

X.E Normalization of vectors

Vectors that are constructed from the sum of components are not likely to have a Euclidean length of one. This causes problems if these results are stored in memory for later similarity matching – a large vector can have a large dot-product with another vector even when the proportion of their shared components is relatively small. In this example this problem was dealt with by including constant factors designed to make the expected length of the result equal to one. This only works when the pair-wise expected similarities of the vectors in the sum are all zero. It is probably preferable to normalize all vectors so that their lengths are exactly one. This was not done in this example because doing so would affect the validity of the analysis of expectations and variances of dot-products. Another alternative is to use the cosine rather than the dot-product as a measure of similarity, but this also makes analysis more difficult.

X.F The use of thresholds

Fixed thresholds are helpful in the analysis of probability of correct retrieval but they are not very good for determining the result of a similarity match in practice. There two reasons for this:

- The best threshold varies with the composition of the HRR frame (e.g., the number of terms in the HRR).
- The best threshold varies with the particular objects in the HRR. (E.g., as t varied with the Z value in Section X.D.)

Consequently, no single fixed threshold will be appropriate for choosing the winning match in all situations. A simpler scheme than using variable thresholds is to choose the most similar match. The situation where there is no filler of a role can be detected with a low

[18] $P' \leq P$ because it can be the case that $Z + X_{mark} < t$ and $Z + X_{mark} > Z + Y_p \ \forall \ p \neq \mathbf{mark}$.

[19] t is chosen with knowledge of Z but not of X_{mark} or Y_p.

threshold. An alternative is to have a no-decision region; if the highest score is not more than some fixed amount greater than the next highest score the result is considered unclear and there is no decision.

XI Discussion

Circular convolution is a bilinear operation, and one consequence of the linearity is low storage efficiency. However, the storage efficiency is sufficient to be usable and scales almost linearly. Convolution is endowed with several positive features by virtue of its linear properties. One is that it preserves similarity - HRRs with similar fillers in similar roles are similar. Another is that convolution can be computed very quickly using FFTs. Another is that analysis of the capacity, scaling, and generalization properties is straightforward. Yet another is that there is potential for a system using HRRs to retain ambiguity while processing ambiguous input.

Convolution can be used as a fixed mapping in a connectionist network to replace one or more of the usual weight-matrix mappings. The forward-propagation of activations and the back-propagation of gradients both can be calculated very quickly using FFTs. This possibility is pursued in [25].

It is possible to do all the calculations of HRRs entirely within the frequency domain. If all vectors were represented in the frequency domain it would not be necessary to do any FFTs and all the operations of HRRs could be done in $O(n)$ time. The rest of the system, including clean-up memories, would have to be able to work with complex vectors. There has been some research on creating adapting neural network architectures to work with units with complex valued activations (e.g., [2]).

One of the problems with convolution memories is the noisy results they give. The noise can be reduced if the encoding vectors have uniform power in the frequency domain. Under this condition the approximate inverse is equal to the exact inverse. Whether or not the advantages afforded by this outweigh the disadvantages of having another constraint is an open question. It is possible to have a HRR system in which all vectors conform to this constraint. In such a system the convolution, involution and dot-product operations are all straightforward but the analogue of addition operation cannot be linear. The properties of such a system remain a subject for investigation.

XII Conclusion

Memory models using circular convolution provide a way of representing compositional structure in distributed representations. They implement Hinton's [12] suggestion that reduced descriptions should have microfeatures that are systematically related to those of their constituents. The operations involved are mostly linear and the properties of the scheme are relatively easy to analyze, especially compared to schemes such Pollack's RAAMs [27]. There is no learning entailed and the scheme works with a wide range of vectors. Systems employing HRRs must have an error-correcting auto-associative memory to clean up the noisy results produced by convolution decoding.

Acknowledgements

Conversations with Jordan Pollack, Janet Metcalfe, and Geoff Hinton have been essential to the development of the ideas expressed in this paper. Comments from Chris Williams and two anonymous reviewers have helped to greatly improve the exposition in this paper. This research has been supported in part by the Canadian Natural Sciences and Engineering Research Council.

Object to extract	Expression	Similarity scores (dot product)		
(1) Agent of $\tilde{\mathbf{s}}_1$	$\tilde{\mathbf{s}}_1 \circledast \mathbf{agt}_{eat}{}^*$	**mark** (0.62)	**john** (0.47)	**paul** (0.41)
(2) Agent of $\tilde{\mathbf{s}}_1$	$\tilde{\mathbf{s}}_1 \circledast \mathbf{agt}^*$	**mark** (0.40)	**john** (0.34)	**person** (0.30)
(3) Object of $\tilde{\mathbf{s}}_1$	$\tilde{\mathbf{s}}_1 \circledast \mathbf{obj}_{love}{}^*$	**the_fish** (0.69)	**fish** (0.44)	**food** (0.39)
(4) Agent of $\tilde{\mathbf{s}}_2$	$\tilde{\mathbf{s}}_2 \circledast \mathbf{agt}_{cause}{}^*$	**hunger** (0.50)	**state** (0.39)	**thirst** (0.31)
(5) Object of $\tilde{\mathbf{s}}_2$	$\tilde{\mathbf{s}}_2 \circledast \mathbf{obj}_{cause}{}^*$	$\tilde{\mathbf{s}}_1$ (0.63)	$\tilde{\mathbf{s}}_3$ (0.46)	**eat** (0.43)
(6) Agent of object of $\tilde{\mathbf{s}}_2$	$\tilde{\mathbf{s}}_2 \circledast \mathbf{obj}_{cause}{}^* \circledast \mathbf{agt}_{eat}{}^*$	**mark** (0.27)	**paul** (0.23)	**luke** (0.22)
(7) Object of object of $\tilde{\mathbf{s}}_2$	$\tilde{\mathbf{s}}_2 \circledast \mathbf{obj}_{cause}{}^* \circledast \mathbf{obj}_{eat}{}^*$	**the_fish** (0.39)	**fish** (0.24)	**food** (0.23)
(8) Object of $\tilde{\mathbf{s}}_3$	$\tilde{\mathbf{s}}_3 \circledast \mathbf{obj}_{eat}{}^*$	**food** (0.07)	**the_bread** (0.06)	**mark** (0.06)
(9) Role of **john** in $\tilde{\mathbf{s}}_4$	$\tilde{\mathbf{s}}_4 \circledast \mathbf{john}^*$	$\mathbf{agt}_{see}$ (0.66)	**agt** (0.50)	$\mathbf{obj}_{see}$ (0.47)
(10)Role of **john** in $\tilde{\mathbf{s}}_5$	$\tilde{\mathbf{s}}_5 \circledast \mathbf{john}^*$	$\mathbf{agt}_{see}$ (0.58)	**agt** (0.41)	$\mathbf{agt}_{eat}$ (0.36)

Table 8: Results of extracting fillers from the frames. In all cases shown the item most similar to the result is the correct one. The similarity comparisons are all with the entire set of features, tokens, roles, and frames. See the text for discussion of each row.

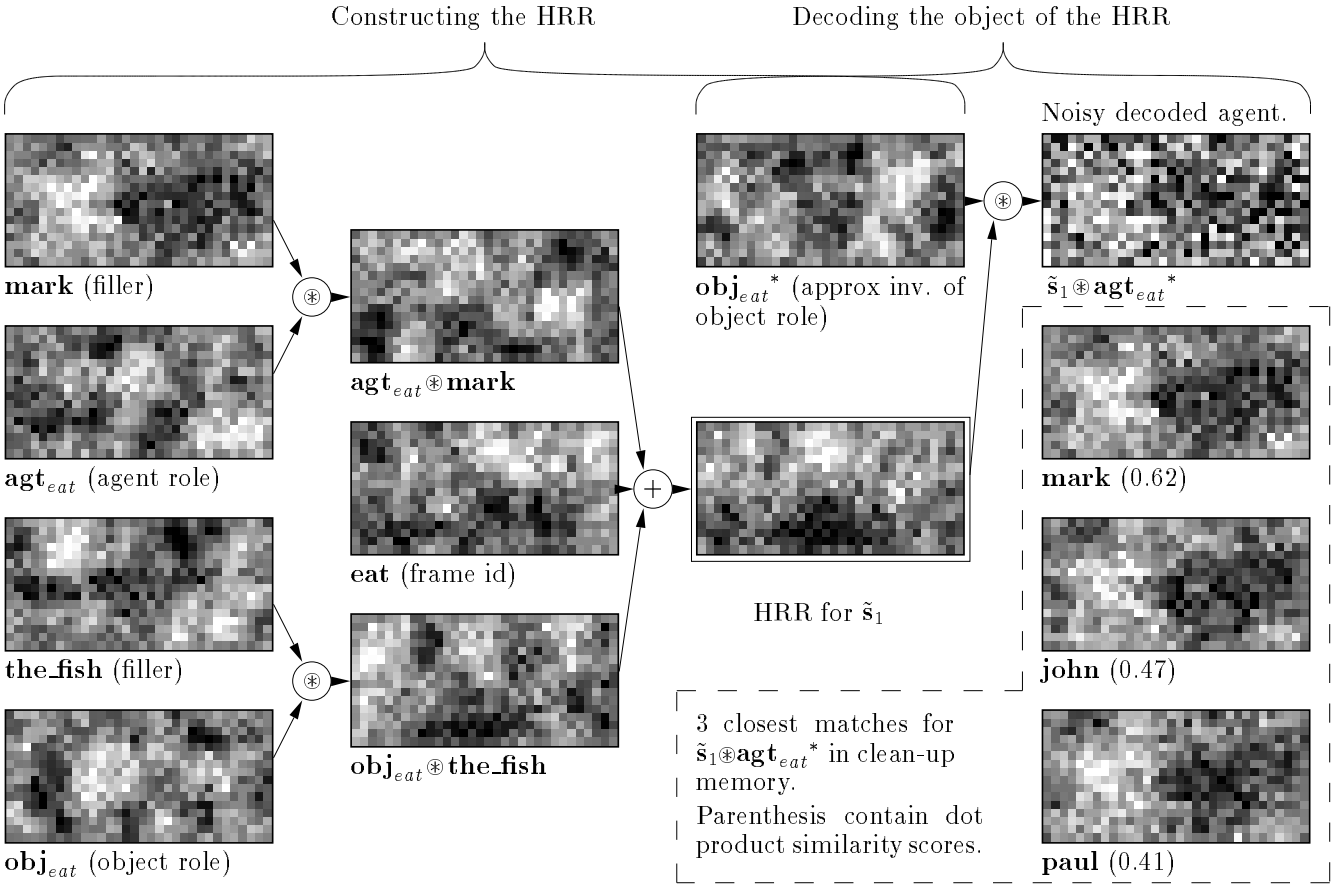


Figure 10: Construction and decoding of a HRR for the sentence "Mark ate the fish." ($\tilde{\mathbf{s}}_1$ in Section X). The instantiated frame, labeled $\tilde{\mathbf{s}}_1$, is the sum of role/filler bindings and a frame id (shown in the second column). It is the same dimensionality as all other objects and may be used a filler in another frame (e.g., as in $\tilde{\mathbf{s}}_2$ in Section X). A filler of the HRR can be extracted by convolving the HRR with the approximate inverse of its role. The extraction of the agent role filler of this sentence is shown on the right (also see Table 8). Of the items in clean-up memory, the actual filler, **mark**, is the most similar (shown in the dotted region). The next two most similar items are also shown, with the dot-product match value in parenthesis. In this high-dimensional space, these two items are significantly less similar than the actual filler. See Section X.C for discussion.

Appendix A. A lower bound for the capacity of addition memories

An addition memory can store a small set of vectors in a single trace. It is easy to recognize whether or not a vector has been stored in a trace. In this appendix I show how the probability of correct recognition is related to the vector dimension and the number of vectors stored. Suppose we have an addition memory trace with the following parameters:

- k distinct items (vectors) stored in a memory trace, selected from m possible vectors, $\tilde{\mathbf{a}}, \tilde{\mathbf{b}}, \tilde{\mathbf{c}}, \tilde{\mathbf{d}}$ etc.
- n elements in each vector, each element (e.g., a_i) independently drawn from a $\mathrm{N}(0, \frac{1}{n})$ distribution.
- q, the probability of making one or more errors while determining which items are (and are not) stored in the memory trace.
- s_a and s_r, the accept and reject signals (see below).

To test whether some item $\tilde{\mathbf{x}}$ is in a trace $\tilde{\mathbf{t}}$, we compute the dot product of $\tilde{\mathbf{x}}$ and $\tilde{\mathbf{t}}$. The resulting signal will be from one of two distributions; the accept distribution S_a (if $\tilde{\mathbf{x}}$ is in the trace), or the reject distribution S_r (if $\tilde{\mathbf{x}}$ is not in the trace). The means and variances of these distributions can be calculated by expanding $\tilde{\mathbf{x}} \cdot \tilde{\mathbf{t}}$. For example, consider the trace $\tilde{\mathbf{t}} = \tilde{\mathbf{a}} + \tilde{\mathbf{b}} + \tilde{\mathbf{c}}$, and a signal from the accept distribution:

$$s_a = \tilde{\mathbf{a}} \cdot \tilde{\mathbf{t}} = \tilde{\mathbf{a}} \cdot \tilde{\mathbf{a}} + \tilde{\mathbf{a}} \cdot \tilde{\mathbf{b}} + \tilde{\mathbf{a}} \cdot \tilde{\mathbf{c}}.$$

Recall that vector elements are distributed as $\mathrm{N}(0, \frac{1}{n})$, from which it follows that $\mathrm{E}(a_i^2) = \frac{1}{n}$ and $\mathrm{var}(a_i^2) = \frac{2}{n^2}$, and $\mathrm{E}(a_i b_i) = 0$ and $\mathrm{var}(a_i b_i) = \frac{1}{n^2}$. By the central limit theorem the terms like $\tilde{\mathbf{a}} \cdot \tilde{\mathbf{a}}$ are distributed as $\mathrm{N}(1, \frac{2}{n})$, and the terms like $\tilde{\mathbf{a}} \cdot \tilde{\mathbf{b}}$ are distributed as $\mathrm{N}(0, \frac{1}{n})$. Since these terms all have zero covariance, we can add means and variances to get $s_a \stackrel{d}{=} \mathrm{N}(1, \frac{k+1}{n})$ and $s_r \stackrel{d}{=} \mathrm{N}(0, \frac{k}{n})$.

If the signal $\tilde{\mathbf{x}} \cdot \tilde{\mathbf{t}}$ is greater than some threshold t we assume that it is from the accept distribution and thus the item is in the trace, and if it is less we assume it is not. This decision procedure is not infallible, but n can be chosen make the probability of error acceptably low.

Using cumulative distribution functions, we can work out the probability $\Pr(\text{Hit}) (= \Pr(s_a > t))$ of correctly deciding an item was stored in a trace, and the probability $\Pr(\text{Reject}) (= \Pr(s_r < t))$ of correctly deciding an item was not stored in a trace. The threshold t can be chosen to maximize the probability $\Pr(\text{Correct})$ of correctly identifying all the items stored (and not stored) in a particular trace:

$$\Pr(\text{Correct}) = \Pr(\text{Hit})^k \Pr(\text{Reject})^{m-k}$$

The probability density functions (pdfs) for s_a and s_r and the optimal single threshold are shown in Figure A.11, for an example with $n = 64$, $m = 100$, and $k = 3$ (for which $\Pr(\text{Correct}) = 0.68$). Note that the optimal scheme for deciding whether a signal comes from the accept or the reject distributions involves testing whether the signal is in a region around the distribution with the smaller variance. For the purposes here the small gains this scheme makes over a single threshold scheme are outweighed by its added complexity.

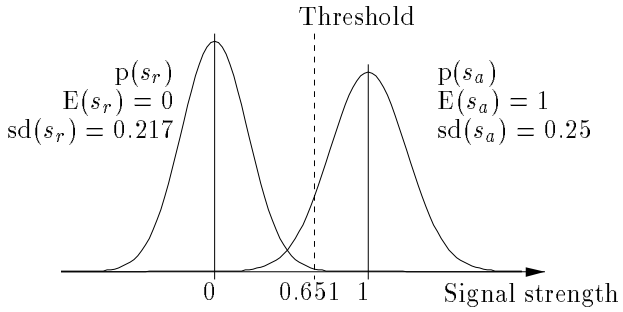


Figure A.11: Distribution of accept (s_a) and reject (s_r) signals for recognition in a linear addition memory, with $n = 64$ and $k = 3$. The threshold shown maximizes $\Pr(\text{Correct})$ for $m = 100$.

It is difficult to find an analytic expression for the capacity, relating k to n, m, and q. However, a reasonably close lower bound can be found as follows. First, some definitions; $\mathrm{erfc}(x)$ is the standard "error function" and $\mathrm{tail}(x)$ is the area under the (normalized) normal probability density function beyond x (in one tail):

$$\mathrm{erfc}(x) = \frac{2}{\pi} \int_x^\infty e^{-t^2} \mathrm{d}t$$

$$\mathrm{tail}(x) = \frac{1}{\sqrt{2\pi}} \int_x^\infty e^{\frac{-t^2}{2}} \mathrm{d}t = \frac{1}{2} \mathrm{erfc}\left(\frac{x}{\sqrt{2}}\right)$$

The following inequality from Abramowitz and Stegun [1] and a simplification are used:

$$\mathrm{erfc}(x) < \frac{2}{\sqrt{\pi}} e^{-x^2} \frac{1}{x + \sqrt{x^2 + \frac{4}{\pi}}}$$

$$\mathrm{erfc}(x) < \frac{1}{x\sqrt{\pi}} e^{-x^2} \tag{1}$$

The probability of correctly identifying all items in the trace can first simplified by chosing $t = 0.5$ and then by applying the binomial theorem:

$$\Pr(\text{Correct}) = \max_t \Pr(a > t)^k \Pr(r < t)^{m-k} \tag{2}$$

$$> \Pr(a > 0.5)^k \Pr(r < 0.5)^{m-k}$$

$$= (1 - \Pr(a < 0.5))^k (1 - \Pr(r > 0.5))^{m-k}$$

$$> 1 - k \Pr(a < 0.5) - (m - k) \Pr(r > 0.5) \tag{3}$$

Now consider q, the probability of one or more errors. This can be simplified by first using Inequality 3 to give Inequality 4, then next replacing smaller variances with the maximum variance to give Inequality 5. After that we use Inequality 1 to give Inequality 6 and finally replace the square root factor by one to give Inequality 7, since it is safe to assume that that factor is less than one.

$$q = 1 - \Pr(\text{Correct})$$

$$< k \Pr(a < 0.5) + (m - k) \Pr(r > 0.5) \tag{4}$$

$$
\begin{aligned}
&= k \text{ tail}\left(\frac{1}{2}\sqrt{\frac{n}{k+1}}\right) + (m-k)\text{tail}\left(\frac{1}{2}\sqrt{\frac{n}{k}}\right) \\
&< k \text{ tail}\left(\frac{1}{2}\sqrt{\frac{n}{k+1}}\right) + (m-k)\text{tail}\left(\frac{1}{2}\sqrt{\frac{n}{k+1}}\right) \quad (5) \\
&= m \text{ tail}\left(\frac{1}{2}\sqrt{\frac{n}{k+1}}\right) \\
&= \frac{m}{2}\text{erfc}\left(\frac{1}{2}\sqrt{\frac{n}{2(k+1)}}\right) \\
&< \frac{m}{\sqrt{\pi}}\sqrt{\frac{2(k+1)}{n}}\; e^{\frac{-n}{8(k+1)}} \quad (6) \\
&< m\, e^{\frac{-n}{8(k+1)}} \qquad \text{if } \sqrt{\frac{2(k+1)}{\pi n}} < 1 \quad (7)
\end{aligned}
$$

Rearranging gives:

$$
\begin{aligned}
n &< 8(k+1)\ln(\frac{m}{q}) \qquad \text{if } n > \frac{2(k+1)}{\pi} \\
\text{or} \quad k &> \frac{n}{8\ln(m/q)} - 1 \qquad \text{if } k < \frac{n\pi}{2} - 1
\end{aligned}
$$

This lower bound on the capacity (k) is reasonably close. Numerical solutions of the exact expression for Pr(Correct) (Equation 2) for k in the range (2..14), m in (10^2...10^{10}), and q in (10^{-2}...10^{-10}) are reasonably well approximated by

$$n = 3.16(k - 0.25)\ln\frac{m}{q^3}.$$

The analysis here treats signal values as random variables, but their randomness is only a consequence of the random choice of the original vectors. For any particular trace with a particular set of vectors, the signal values are deterministic. This style of analysis is consistently used throughout this paper: there are no stochastic operations, only randomly chosen vectors.

Appendix B. A lower bound for the capacity of convolution memories

In this appendix I show how the analysis in Appendix A can be extended to a convolution memory that stores pairs of items. Instead of storing k items in a trace, we store k pairs of items.[20] Parameters n, m, and q are as described at the beginning of Appendix A. We can check whether a cue-probe pair has been stored in the trace by first convolving the trace with the approximate inverse of the cue and then checking the similarity (dot-product) to the probe.[21] I assume that we do not know what the appropriate cues are, so to find all the pairs in the trace we must try every combination of cue and probe.

There are five distributions of reject signals and one distribution of accept signals. The distribution of a reject signal depends on how many of the cue and probe occurred in the trace (0, 1, or 2) and on whether the cue was equal to the probe. The signals are summarized in Table B.9, with examples for the trace $\tilde{\mathbf{t}} = \tilde{\mathbf{a}} \circledast \tilde{\mathbf{b}} + \tilde{\mathbf{c}} \circledast \tilde{\mathbf{d}}$. The means and variances of all the signals are also shown, together with the number of each of these signals that must be tested to probe exhaustively for each pair in the trace. The variances were calculated by adding the appropriate variances from Table 1, ignoring the terms of order $1/n^2$. The covariance between all terms in the signals (e.g., $\tilde{\mathbf{a}} \circledast \tilde{\mathbf{b}} \circledast \tilde{\mathbf{a}}^* \cdot \tilde{\mathbf{e}}$ and $\tilde{\mathbf{c}} \circledast \tilde{\mathbf{d}} \circledast \tilde{\mathbf{a}}^* \cdot \tilde{\mathbf{e}}$) is zero.[22]

Signal	(Example)	E	var	Number of tests
a	$(\tilde{\mathbf{t}} \circledast \tilde{\mathbf{a}}^* \cdot \tilde{\mathbf{b}})$	1	$\frac{k+5}{n}$	k
$r_{1=}$	$(\tilde{\mathbf{t}} \circledast \tilde{\mathbf{a}}^* \cdot \tilde{\mathbf{a}})$	0	$\frac{2k+4}{n}$	$2k$
r_2	$(\tilde{\mathbf{t}} \circledast \tilde{\mathbf{a}}^* \cdot \tilde{\mathbf{c}})$	0	$\frac{k+2}{n}$	$2k(k-1)$
r_1	$(\tilde{\mathbf{t}} \circledast \tilde{\mathbf{a}}^* \cdot \tilde{\mathbf{e}})$	0	$\frac{k+1}{n}$	$2k(m-2k)$
$r_{0=}$	$(\tilde{\mathbf{t}} \circledast \tilde{\mathbf{e}}^* \cdot \tilde{\mathbf{e}})$	0	$\frac{2k}{n}$	$m-2k$
r_0	$(\tilde{\mathbf{t}} \circledast \tilde{\mathbf{e}}^* \cdot \tilde{\mathbf{f}})$	0	$\frac{k}{n}$	$\frac{(m-2k)(m-2k-1)}{2}$

Table B.9: Means and variances of signals in a paired associates convolution memory.

The probability of correctly identifying all the pairs in the trace is:[23]

$$
\begin{aligned}
\Pr(\text{Correct}) &= \max_t \Pr(a > t)^k \\
&+ \Pr(r_{1=} < t)^{2k} + \Pr(r_2 < t)^{2k(k-1)} \\
&+ \Pr(r_1 < t)^{2k(n-2k)} + \Pr(r_{0=} < t)^{n-2k} \\
&+ \Pr(r_0 < t)^{(n-2k)(n-2k-1)/2}. \qquad (8)
\end{aligned}
$$

Using the same inequalities as in Appendix A we get:

$$
\begin{aligned}
q &< \frac{m(m+1)}{2}\text{ tail}(\frac{1}{2}\sqrt{\frac{2k+4}{n}}) \\
&< m^2\, e^{\frac{-n}{16(k+2)}} \qquad \text{if } n > \frac{4(k+2)}{\pi} \\
\text{or} \quad n &< 16(k+2)\ln\frac{m^2}{q} \qquad \text{if } n > \frac{4(k+2)}{\pi}
\end{aligned}
$$

Numerical solutions of the Equation 8 for k in the range (2..14), m in (10^2...10^{10}), and q in (10^{-2}...10^{-10}) are reasonably well approximated by

$$n = 4.5(k + 0.7)\ln\frac{m}{30q^4}.$$

Appendix C: Means and variances of a signal

The detailed calculation of the mean and variance one of the components of one of the signals in Section X.D is presented in this appendix.

[20] Some complications are avoided by not permitting repetitions or pairs of identical items.

[21] I do not assume that the retrieval process avoids checking whether the probe and cue are equal.

[22] If we had allowed identical pairs in the trace this would not be true for all signals, since $\tilde{\mathbf{b}} \circledast \tilde{\mathbf{b}} \circledast \tilde{\mathbf{a}}^* \cdot \tilde{\mathbf{e}}$ and $\tilde{\mathbf{c}} \circledast \tilde{\mathbf{c}} \circledast \tilde{\mathbf{a}}^* \cdot \tilde{\mathbf{e}}$ have non-zero covariance.

[23] This expression actually underestimates Pr(Correct) because a small proportion signal values are correlated with each other. This causes clustering of errors – if a trace has at least one decoding error it is likely to have more.

Expanding the signal X_{mark} gives

$$\begin{aligned}
X_{mark} &= \tilde{\mathbf{s}}_1 \circledast \mathbf{agt}_{eat}{}^* \cdot \frac{1}{\sqrt{3}}\mathbf{id}_{mark} \\
&= \frac{1}{\sqrt{3}}(\mathbf{eat} + \mathbf{agt}_{eat} \circledast \mathbf{mark} \\
&= +\mathbf{obj}_{eat} \circledast \mathbf{the_fish}) \circledast \mathbf{agt}_{eat}{}^* \cdot \frac{1}{\sqrt{3}}\mathbf{id}_{mark} \\
&= \frac{1}{3}(\mathbf{eat} \\
&= +\mathbf{agt}_{eat} \circledast \frac{1}{\sqrt{3}}(\mathbf{being} + \mathbf{person} + \mathbf{id}_{mark}) \\
&\quad +\mathbf{obj}_{eat} \circledast \mathbf{the_fish}) \circledast \mathbf{agt}_{eat}{}^* \cdot \mathbf{id}_{mark} \\
&= \frac{1}{3}\mathbf{eat} \circledast \mathbf{agt}_{eat}{}^* \cdot \mathbf{id}_{mark} \\
&\quad +\frac{1}{3\sqrt{3}}\mathbf{agt}_{eat} \circledast \mathbf{being} \circledast \mathbf{agt}_{eat}{}^* \cdot \mathbf{id}_{mark} \\
&\quad +\frac{1}{3\sqrt{3}}\mathbf{agt}_{eat} \circledast \mathbf{person} \circledast \mathbf{agt}_{eat}{}^* \cdot \mathbf{id}_{mark} \\
&\quad +\frac{1}{3\sqrt{3}}\mathbf{agt}_{eat} \circledast \mathbf{id}_{mark} \circledast \mathbf{agt}_{eat}{}^* \cdot \mathbf{id}_{mark} \\
&\quad +\frac{1}{3}\mathbf{obj}_{eat} \circledast \mathbf{the_fish} \circledast \mathbf{agt}_{eat}{}^* \cdot \mathbf{id}_{mark}
\end{aligned}$$

The expectations and variances of these terms can be found by consulting Table 1. It is not necessary to expand the vectors $\mathbf{agt}_{eat}$, $\mathbf{obj}_{eat}$ or $\mathbf{the\_fish}$, as the components of these are independent of the other vectors appearing in the same terms. The expectation of the fourth term is $\frac{1}{3\sqrt{3}}$ (row 4 in Table 1), and all the expectation of the remaining terms is zero. These five terms are independent and thus the variance of the sum is the sum of the variances. The variance of the first term is $\frac{1}{9n}$ (row 3 in Table 1), the variance of the second and third terms is $\frac{2n+2}{27n^2}$ (row 8), the variance of the fourth term is $\frac{6n+4}{27n^2}$ (row 6), and the variance of the fifth term is $\frac{1}{9n}$ (row 10). These terms are uncorrelated, so their expectations and variances can be summed to give

$$\mathrm{E}(X_{mark}) = \frac{1}{3\sqrt{3}}, \quad \mathrm{var}(X_{mark}) = \frac{16n+8}{27n^2} \approx 0.593/n.$$

The expectations and variances of Y_p and Z can be calculated in a similar manner. They are:

$$\mathrm{E}(Y_p) = 0, \quad \mathrm{var}(Y_p) = \frac{12n+6}{27n^2} \approx 0.444/n$$

$$\mathrm{E}(Z) = \frac{2}{3\sqrt{3}}, \quad \mathrm{var}(Z) = \frac{34n+20}{27n^2} \approx 1.26/n$$

References

[1] Milton Abramowitz and Irene A. Stegun, editors. *Handbook of mathematical functions with formulas, graphs, and mathematical tables*. Dover, New York, 1965.

[2] N. Benvenuto and F. Piazza. On the Complex Backpropagation Algorithm. *IEEE Transactions on Signal Processing*, 40(4):967–969, 1992.

[3] A. Borsellino and T. Poggio. Convolution and correlation algebras. *Kybernetik*, 13:113–122, 1973.

[4] E. O. Brigham. *The Fast Fourier Transform*. Prentice Hall, Inc., New Jersey, 1974.

[5] David Casasent and Brian Telfer. Key and recollection vector effects on heteroassociative memory performance. *Applied Optics*, 28(2):272–283, 1989.

[6] Phillip J. Davis. *Circulant matrices*. John Wiley & Sons, New York, 1979.

[7] Douglas F. Elliot. *Handbook of Digital Signal Processing Engineering Applications*. Academic Press, Inc, San Diego, CA, 1986.

[8] Arthur D. Fisher, Wendy L. Lippincott, and John N. Lee. Optical implementations of associative networks with versatile adaptive learning capabilities. *Applied Optics*, 26(23):5039–5054, 1987.

[9] J. A. Fodor and Z. W. Pylyshyn. Connectionism and cognitive architecture: A critical analysis. *Cognition*, 28:3–71, 1988.

[10] Robert A. Gabel and Richard A. Roberts. *Signals and Linear systems*. John Wiley & Sons, Inc, 1973.

[11] G. E. Hinton. Implementing semantic networks in parallel hardware. In G. E. Hinton and J. A. Anderson, editors, *Parallel Models of Associative Memory*. Hillsdale, NJ: Erlbaum, 1981.

[12] G. E. Hinton. Mapping part-whole heirarchies into connectionist networks. *Artificial Intelligence*, 46(1-2):47–76, 1990.

[13] G. E. Hinton, J. L. McClelland, and D. E. Rumelhart. Distributed representations. In J. L. McClelland D. E. Rumelhart and the PDP research group, editors, *Parallel distributed processing: Explorations in the microstructure of cognition.*, volume I, pages 77–109. Cambridge, MA: MIT Press, 1986.

[14] J. J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences U.S.A.*, 79:2554–2558, 1982.

[15] P. Kanerva. *Sparse Distributed Memory*. MIT Press, Cambridge, MA, 1988.

[16] G. Legendre, Y. Miyata, and P. Smolensky. Principles for an Integrated Connectionist/Symbolic Theory of Higher Cognition. Technical Report CU-CS-600-92, University of Colorado at Boulder, 1992.

[17] Stephan Lewandowsky and Bennet B. Murdock. Memory for serial order. *Psychological Review*, 96(1):25–57, 1989.

[18] Janet Metcalfe Eich. A composite holographic associative recall model. *Psychological Review*, 89:627–661, 1982.

[19] B. B. Murdock. A distributed memory model for serial-order information. *Psychological Review*, 90(4):316–338, 1983.

[20] Bennet B. Murdock. A theory for the storage and retrieval of item and associative information. *Psychological Review*, 89(6):316–338, 1982.

[21] Bennet B. Murdock. Serial-order effects in a distributed-memory model. In David S. Gorfein and Robert R. Hoffman, editors, *MEMORY AND LEARNING: The Ebbinghaus Centennial Conference*, pages 277–310. Lawrence Erlbaum Associates, 1987.

[22] Eung Gi Paek and Demetri Psaltis. Optical associative memory using Fourier transform holograms. *Optical Engineering*, 26(5):428–433, 1987.

[23] Ray Pike. Comparison of convolution and matrix distributed memory systems for associative recall and recognition. *Psychological Review*, 91(3):281–294, 1984.

[24] Tony A. Plate. Holographic Reduced Representations. Technical Report CRG-TR-91-1, Department of Computer Science, University of Toronto, 1991.

[25] Tony A. Plate. Holographic recurrent networks. In C. L. Giles, S. J. Hanson, and J. D. Cowan, editors, *Advances in Neural Information Processing Systems 5 (NIPS*92)*, pages 34–41, San Mateo, CA, 1992. Morgan Kaufmann.

[26] T. Poggio. On holographic models of memory. *Kybernetik*, 12:237–238, 1973.

[27] J. B. Pollack. Recursive distributed representations. *Artificial Intelligence*, 46(1-2):77–105, 1990.

[28] R. Rosenfeld and D. S. Touretzky. Coarse-coded symbol memories and their properties. *Complex Systems*, 2(4):463–484, 1988.

[29] D. E. Rumelhart, G. E. Hinton, and Williams R. J. Learning internal representations by error propagation. In J. L. McClelland D. E. Rumelhart and the PDP research group, editors, *Parallel distributed processing: Explorations in the microstructure of cognition*, volume I. Bradford Books, Cambridge, MA, 1986.

[30] P. H. Schönemann. Some algebraic relations between involutions, convolutions, and correlations, with applications to holographic memories. *Biological Cybernetics*, 56:367–374, 1987.

[31] J. N. Slack. A parsing architecture based on distributed memory machines. In *Proceedings of COLING-86*, pages 476–481. Association for Computational Linguistics, 1986.

[32] P. Smolensky. Tensor product variable binding and the representation of symbolic structures in connectionist systems. *Artificial Intelligence*, 46(1-2):159–216, 1990.

[33] D. S. Touretzky. BoltzCONS: Dynamic symbol structures in a connectionist network. *Artificial Intelligence*, 42(1-2):5–46, 1990.

[34] D. S. Touretzky and S. Geva. A distributed connectionist representation for concept structures. In *Proceedings of the Ninth Annual Cognitive Science Society Conference*, Hillsdale, NJ, 1987. Erlbaum.

[35] D. S. Touretzky and G. E. Hinton. A distributed connectionist production system. *Cognitive Science*, 12(3):423–466, 1988.

[36] Elke U. Weber. Expectation and variance of item resemblance distributions in a convolution-correlation model of distributed memory. *Journal of Mathematical Psychology*, 32:1–43, 1988.

[37] D. Willshaw. Holography, associative memory, and inductive generalization. In G. E. Hinton and J. A. Anderson, editors, *Parallel models of associative memory*. Erlbaum, Hillsdale, NJ, 1981.

[38] D. Willshaw and P. Dayan. Optimal plasticity from matrix memories: What goes up must come down, 1990.