Holographic Reduced Representations: Convolution Algebra for Compositional Distributed Representations

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

Abstract

A solution to the problem of representing compositional structure using distributed representations is described. The method uses circular convolution to associate items, which are represented by vectors. Arbitrary variable bindings, short sequences of various lengths, frames, and reduced representations can be compressed into a fixed width vector. These representations are items in their own right, and can be used in constructing compositional structures. The noisy reconstructions given by convolution memories can be cleaned up by using a separate associative memory that has good reconstructive properties.

1 Introduction

Distributed representations [Hinton, 1984] 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 in distributed representations has been for some time a prominent concern of both followers and critics of the connectionist faith [Fodor and Pylyshyn, 1988; Hinton, 1990] .

Using connectionist networks, e.g., back propagation nets, Hopfield nets, Boltzrnann machines, or Willshaw nets, it is easy to represent associations of a fixed number of items. The difficulty with representing compositional structure in all of these networks is that items and associations are represented in different spaces. Hinton [1990] discusses this problem and proposes a framework in which "reduced descriptions" are used. This framework requires that a number of vectors be compressed (reduced) into a single vector of the same size as each of the original vectors. This vector acts as a reduced description of the set of vectors and itself can be a member of another set of vectors. The reduction must be reversible so that one can move in both directions in a part-whole hierarchy. In this way, compositional structure is represented. However, Hinton does not suggest any concrete way of performing this reducing mapping.

Some researchers have built models or designed frameworks in which some some compositional structure is present in distributed representations. For some examples see the papers of Touretzky, Pollack, or Smolensky in [AIJ, 1990].

In this paper I propose a new method for representing compositional structure in distributed representations. Circular convolution is 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 representations of objects with compositional structure. 1 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 [1990] reduced descriptions. 1 describe how HRRs and error correcting 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.

2 Associative memories

Associative memories are used to store associations between items which are represented is a distributed fashion as vectors. Nearly all work on associative memory has been concerned with storing items or pairs of items.

Convolution-correlation memories (sometimes refered to as holographic-like) and matrix memories have been regarded as alternate methods for implementing associative memory [Willshaw, 1981; Murdock, 1983; Pike, 1984; Schonemann, 1987]. Matrix memories have received more interest, probably due to their relative simplicity and their higher capacity in terms of the number of elements in the items being associated.

The properties of matrix memories are well understood. Two of the best known matrix memories are "Willshaw" networks [Willshaw, 1981] and Hopfield networks.[Hopfield, 1982] Matrix memories can be used to construct auto-associative (or "content addressable") memories for pattern correction and completion. They can also be used to represent associations between two vectors. After two vectors are associated one can be used as a cue to retrieve the other.

There are three operations used in 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 a noisy version of the item that was originally associated with the cue. Memory traces can be composed (by addition or superposition) and the decoding operation will work with this sum of individual traces, but the retrieved item will be noisier. In some models encoding and decoding are linear, e.g., Murdock [1983], in others decoding is non-linear, e.g., Hopfield [1982], in others all the operations are non-linear, e.g., Willshaw [1981].

To illustrate this, let $\mathbf{I}$ be the space of vectors representing items [1], and $\mathbf{T}$ be the space of vectors or matrices representing memory traces. Let $\boxtimes : \mathbf{I} \times \mathbf{I} \rightarrow \mathbf{T}$ be the encoding operation, $\triangleright$: $\mathbf{I} \times \mathbf{T} \rightarrow \mathbf{I}$ be the decoding operation, and $\boxplus : \mathbf{T} \times \mathbf{T} \rightarrow \mathbf{T}$ be the trace composition operation. Let $\mathbf{a}$, $\mathbf{b}$, $\mathbf{c}$, $\mathbf{d}$, $\mathbf{e}$, and $\mathbf{f}$ be item vectors, and let T_i be memory traces.

The association of two items $\mathbf{a}$ and $\mathbf{b}$ is represented by the trace $T_1 = \mathbf{a} \boxtimes \mathbf{b}$. We can recover $\mathbf{a}$ from T_1 by using the decoding operation on T_1 and the cue $\mathbf{b}$: $\mathbf{b} \triangleright T_1$ is a degraded, or noisy, version of $\mathbf{a}$. Noisy versions of $\mathbf{b}$ can also be used as cues. Depending on the properties of the particular scheme, the retrieved vector will be more or less similar to $\mathbf{a}$.

A trace can represent a number of associations, e.g., $T_2 = (\mathbf{a} \boxtimes \mathbf{b}) \boxplus (\mathbf{c} \boxtimes \mathbf{d}) \boxplus (\mathbf{e} \boxtimes \mathbf{f})$. An item from any pair can be used as a cue to recover the other item of the pair, e.g., $\mathbf{c} \triangleright T_2$ gives a noisy version of $\mathbf{d}$. 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.

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

2.1 Convolution-correlation memories

In nearly all convolution memory models the aperiodic convolution operation has been used to form associations.[2] The aperiodic convolution of two vectors with n elements each 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 resulting vectors grow with recursive convolution. This same growing property is exhibited in a much more dramatic form by both matrix memories and Smolensky's [1990J tensor product representations.

Researchers have used three solutions to this problem of growth with recursive associations - (a) limit the depth of composition (Smolensky [1990]), (b) discard elements outside the n central ones (Metcalfe [1982]), and (c) use infinite vectors (Murdock [1982]).

The growth problem can be avoided entirely by the use of circular convolution, an operation well known in signal processing. The result of the circular convolution of two vectors of *n* elements has just n elements. Since circular convolution does not have the growth property, it can be used recursively in connectionist systems with fixed width vectors.

There is a close relationship between matrix and convolution memories A 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 transdiagonals of the outer product.

Circular convolution, denoted by the symbol $\circledast$, is illustrated in Figure 1. Each of the small circles represents the product of a pair of elements from $\mathbf{a}$ and $\mathbf{b}$, and these are summed along the indicated diagonals. While the circular convolution operation is straightforward, what is remarkable is that circular correlation, $\circledcirc$, (illustrated in Figure 2) is its approximate inverse.[3] If a pair of vectors is convolved together to give a "memory trace", then one of the pair, the "cue", can be used to retrieve the other from the trace. Suppose we have a trace which is the convolution of the cue with another vector, $\mathbf{t} = \mathbf{c} \circledast \mathbf{x}$. Then correlation allows the reconstruction of a noisy version of $\mathbf{x}$ from $\mathbf{t}$ and $\mathbf{c}$: $\mathbf{y} = \mathbf{c} \circledcirc \mathbf{t}$ where $\mathbf{y} \approx \mathbf{x}$. The correlation operation also has aperiodic and circular versions.

Circular convolution is defined as: $\circledast : \mathbf{R}^n \times \mathbf{R}^n \rightarrow \mathbf{R}^n$, such that $\mathbf{t} = \mathbf{c} \circledast \mathbf{x}$ where $t_i = \sum_{j=0}^{n-1} c_j x_{i-j}$. Subscripts are interpreted modulo n, which gives the operation its circular nature.

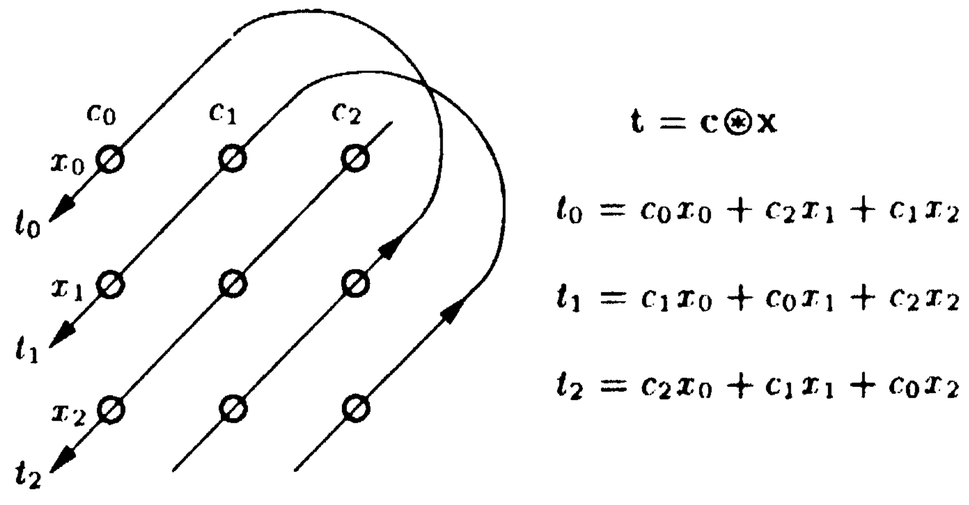

Figure 1: Circular convolution represented as a compressed outer product.

Circular correlation is defined as: $\circledcirc : \mathbf{R}^n \times \mathbf{R}^n \rightarrow \mathbf{R}^n$, such that $\mathbf{y} = \mathbf{c} \circledcirc \mathbf{t}$ where $y_i = \sum_{j=0}^{n-1} c_j t_{j+i}$.

Convolution can be computed in $O(n \log n)$ time using Fast Fourier Transforms (FFTs). The method is simple and well known: $\mathbf{a} \circledast \mathbf{b} = f^{-1}(f\mathbf{a} \odot f\mathbf{b})$, where f is a discrete Fourier transform (its range is a vector of complex numbers), f^{-1} is its inverse, and $\odot$ is the componentwise multiplication of two vectors.

[1] There are usually distributional constraints on the elements of the vectors, e.g., the elements should be drawn from independent distributions.

[2] The exception is the non-linear correlograph of Willshaw [1981], first published in 1969.

[3] Under certain conditions on the distribution of elements in the vectors.

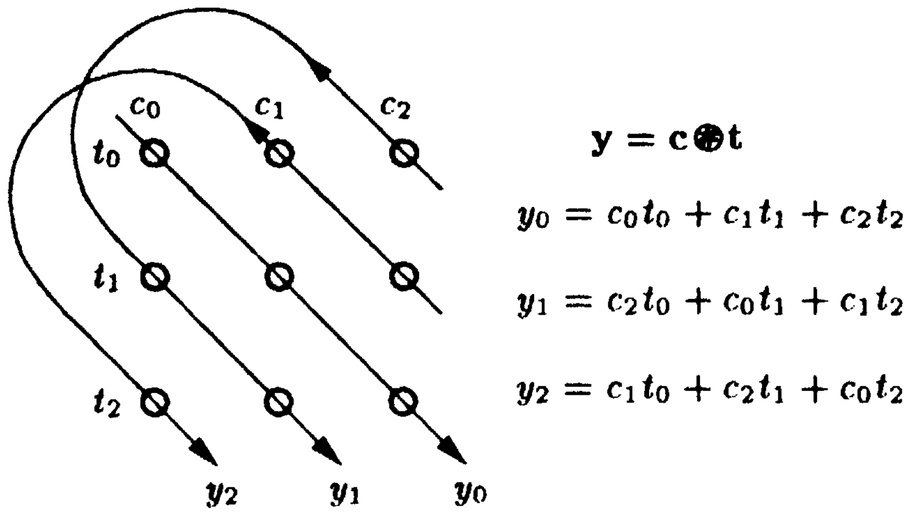

Figure 2: Circular correlation represented as a compressed outer product.

2.2 Distributional constraints on vectors

For correlation to decode convolution the elements of each vector must be independently distributed with mean zero and variance 1/n so that the euclidean length of each vector has a mean of one. Examples of suitable distributions are the normal distribution and the discrete distribution with values equiprobably $\pm 1/\sqrt{n}$. The analysis of signal strength and capacity depends on elements of vectors being independently distributed.

The tension between these constraints and the need for vectors to have meaningful features is discussed in Plate [1991].

2.3 How much information is stored

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; to successfully store a vector 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 $2k \log M$ bits of information are needed to represent *k* pairs of those items.[4] The size of the vectors does not enter into this calculation, only the number of vectors matters.

3 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.

The principle of addition memory can be stated as "adding together two high dimensional vectors gives something which is similar to each and not very similar to anything else."[5] This principle underlies both convolution and matrix memories and the same sort of analysis can be applied to the linear versions of each. Addition memories are discussed at greater length in Plate [I99I]

[4]Slightly less than $2f \log M$ bits are required since the pairs are unordered

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

4 The need for reconstructive item memories

If a system using convolution representations is to do some sort of recall (as opposed to recognition), then it must have an additional error correcting 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 could 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.

For example, suppose the system is to store pairs of letters, and suppose each one of the 26 letters is represented by the random vectors a, b, ...,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 3. The trace is a sum of convolved pairs, e.g., t = a®b -f c®d -f e®f. When the system is given one item as an input cue 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 a as a cue it should produce b and a high strength. When given g as a cue it should give a low strength. The item it outputs is unimportant when the strength is low.

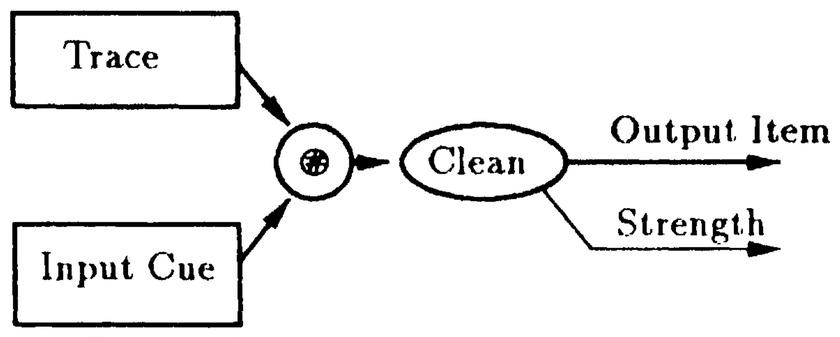

Figure 3: A hetero-associator machine.

The convolution trace stores only a few associations or items, and the item memory stores many items. The item memory acts as an auto-associator to clean up the noisy items retrieved from the convolution trace.

The exact method of implementation of the item memory is unimportant. Hopfield networks are probably not a good candidate because of their low capacity. Kanerva networks [Kanerva, 1988] have sufficient capacity, but can only store binary vectors.[6] For experiments I have been using a nearest neighbor matching memory.

5 Representing more complex structure

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

5.1 Sequences

Sequences can be represented in a number of ways using convolution encoding. An entire sequence can be repre-

Although most of this paper assumes items are represented as real vectors, convolution memories also work with binary vectors [Willshaw, 198l].

sented in one memory trace (providing the soft capacity limits are not exceeded), or chunking can be used to represent a sequence of any length in a number of memory traces.

Murdock [1983; 1987] proposes a chaining method of representing sequences in a single memory trace, and models 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 abc, then the trace is

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

where α_i and β_i are suitable weighting constants less than 1, generated from a two or three underlying parameters, with $\alpha_i > \alpha_{i+1}$. The retrieval of the sequence begins with retrieving the strongest component of the trace, which will be 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.

Another way to represent sequences is to use the entire previous sequence as context rather than just the previous item [Murdock, 1987]. This makes it possible to store sequences with repetitions of items. To store abc, the trace is: $\mathbf{a} + \mathbf{a}\circledast\mathbf{b} + \mathbf{a}\circledast\mathbf{b}\circledast\mathbf{c}$. 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, so to store abc, the trace is: $\mathbf{p}_1\circledast\mathbf{a} + \mathbf{p}_2\circledast\mathbf{b} + \mathbf{p}_3\circledast\mathbf{c}$. The retrieval (and storage) cues p_i can be arbitrary or generated in some manner from a single vector, e.g., $\mathbf{p}_i = (\mathbf{p})^i$.

5.2 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 [Murdock, 1987].

The second sequence representation method is the most suitable one to do chunking with. Suppose we want to represent the sequence abedefgh. We can create three new items representing subsequences:

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

These new items must be added to the item memory and marked in some way as non-terminals. The representation for the whole sequence is:

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

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 7.2.

5.3 Variable binding

It is simple to implement variable binding with convolution: convolve the variable representation with the value representation. If it is desired that the representation of a variable binding should be somewhat similar to both the variable and the value, the vectors for those can be added in. This gives $\alpha\mathbf{x}\circledast\mathbf{a} + \beta\mathbf{x} + \cdot$ as the representation of a variable binding.

This type of variable binding can also be implemented in other types of associative memory, e.g., the triple-space of BoltzCONS [Touretzky and Hinton. 1985], or the outer product of roles and fillers in DUCS [Touretzky and Geva, 1987]. However, in those systems the variable and value objects were of a different dimension than the binding object. Thus it was not possible add components of the variable and the value to the representation of the binding, nor use the binding itself as an item in another association.

5.4 Frame-slot representations

Frames can be represented using convolution encoding in an analogous manner to cross products of roles and fillers in [Hinton, 1981] or the frames of DUCS [Touretzky and Geva, 1987]. 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 "seeing". The vector for the frame label is L_{see} and the vectors for the roles are r_{agent} and r_{object}. This frame could be instantiated with the fillers fj_{ane} and f_{spot} to represent "Jane saw Spot":

$$\mathbf{t}_{seeing} = \mathbf{l}_{see} + \mathbf{r}_{agent}\circledast\mathbf{f}_{jane} + \mathbf{r}_{object}\circledast\mathbf{f}_{spot}$$

Fillers (or roles) can be retrieved from the instantiated frame by correlating with the role (or filler). The vectors representing roles could be frame specific, i.e., r_{agent}-*see* could be different from $\mathbf{r}_{agent-run}$, or they could be the same (or just similar). Uninstantiated frames can also be stored as the sums of the vectors representing their components, e.g., $\mathbf{u}_{seeing} = \mathbf{l}_{see} + \mathbf{r}_{agent} + \mathbf{r}_{object}$. Section 7.1 describes one way of manipulating uninstantiated frames and selecting appropriate roles to fill.

The frame representation can be made similar to any vector by adding some of that vector to it. For example, we could add $\alpha\mathbf{f}_{jane}$ to the above instantiated frame to make the representation for Jane doing something have some similarity to the representation for Jane.

6 Reduced Representations

Using 'he types of representations described in the last Lection, it is a trivial step to building reduced representations which can represent complex hierarchical structure in a fixed width vector. We can use an instantiated frame[7] as a filler instead of f_{spot} in the frame built in the previous section. For example, "Spot ran.":

$$\mathbf{t}_{running} = \mathbf{l}_{run} + \mathbf{r}_{agent}\circledast\mathbf{f}_{spot}$$

[7] Normalization of lengths of vectors becomes an issue, but 1 do not consider this for lack of space.

could be used in a "seeing" frame "Dick saw Spot run.":

$$\begin{aligned} \mathbf{t}_{seeing} &= \mathbf{l}_{see} + \mathbf{r}_{agent} \circledast \mathbf{f}_{dick} + \mathbf{r}_{object} \circledast \mathbf{t}_{running} \\ &= \mathbf{l}_{see} + \mathbf{r}_{agent} \circledast \mathbf{f}_{dick} \\ &\quad + \mathbf{r}_{object} \circledast (\mathbf{l}_{run} + \mathbf{r}_{agent} \circledast \mathbf{f}_{spot}) \end{aligned}$$

This representation can be manipulated with or without chunking. Without chunking, we could extract the agent of the object by correlating with r_{object} ® *ragent-* Using chunking, we could extract the object by correlating with *r0bject,* clean it up, and then extract its agent, giving a less noisy vector than without chunking.

This implements Hinton's idea [1990] of a system being able to focus attention on constituents as well as being able to have the whole meaning present at once. It also suggests the possibility of sacrificing accuracy for speed — if chunks are not cleaned up the retrievals are less accurate.

7 Simple Machines that use HRRs

In this section two simple machines that operate on complex convolution representations are described. Both of these machines have been successfully simulated on a convolution calculator using vectors with 1024 elements.

7.1 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 way of extracting the most appropriate role from an uninstantiated frame. The most appropriate role for a particular filler 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.

Let the uninstantiated frame be $\beta\mathbf{l}+\alpha_1\mathbf{r}_1+\alpha_2\mathbf{r}_2+\alpha_3\mathbf{r}_3$. The task is to select the role that combines best with $\mathbf{f}$, the filler. Suppose there is some item $\mathbf{r}_2 \circledast \mathbf{f}'$ in the item memory, such that $\mathbf{f}'$ is quite similar to $\mathbf{f}$. The presence of a similar binding in the item memory defines $\mathbf{r}_2$ as the "best fitting" role for $\mathbf{f}$.[8]

If the roles in the frame should be selected according to best fit, then the α_i should be approximately equal, but if $\mathbf{r}_1$ 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., $\mathbf{f} \circledast (\beta\mathbf{l} + \alpha_1\mathbf{r}_1 + \alpha_2\mathbf{r}_2 + \alpha_3\mathbf{r}_3)$. This is cleaned up in item memory to give $\mathbf{r}_2 \circledast \mathbf{f}'$, which is then correlated with $\mathbf{f}$ to give a vector which can be written as $\gamma\mathbf{r}_2 + \mathbf{v}_{noise}$ where γ and the noise depend on the similarity of $\mathbf{f}$ to $\mathbf{f}'$.

This result is added to the uninstantiated frame to give $\beta\mathbf{l}+\alpha_1\mathbf{r}_1+(\alpha_2+\gamma)\mathbf{r}_2+\alpha_3\mathbf{r}_3+\mathbf{v}_{noise}$. The strongest role can be selected by cleaning up in item memory. Which is strongest will depend on the relative strengths of α_1, $(\alpha_2 + \gamma)$, and α_3, and the value of γ in turn depends on the similarity of $\mathbf{f}'$ to $\mathbf{f}$.

[8] This form of generalization by similarity can be used extensively.

The machine that accomplishes this operation is shown in Figure 4.

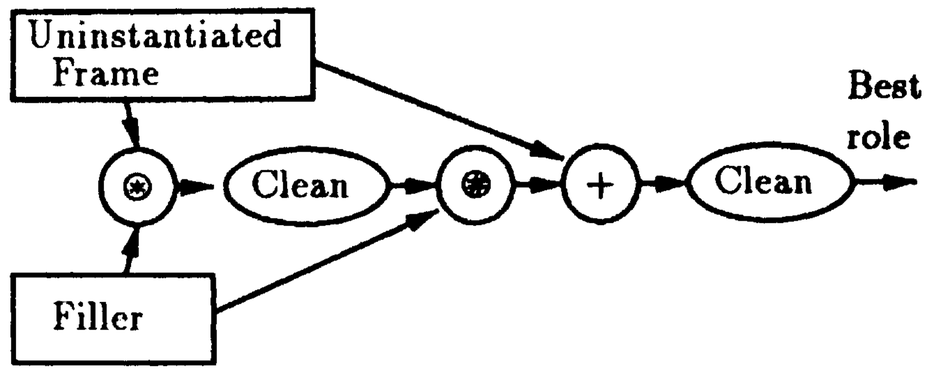

Figure 4: A role selection mechanism

7.2 Chunked sequence readout machine

A machine that reads out the chunked sequences described in section 5.2 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 in [Plate, 1991] , or in a network with fast weights. The machine is shown in Figure 5.

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: x = Clean(t).

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

Non-terminal:

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

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

Pop:

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

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

This machine is an example of a system that can have the whole meaning present and that can also focus attention on constituents.

8 Mathematical properties

Mathematical properties of circular convolution and correlation are discussed in Plate [I99I], including: algebraic properties; the reason convolution is an approximate inverse for correlation; the existence and usefulness of exact inverses of convolutions; and the variances of dot products of various convolution products.

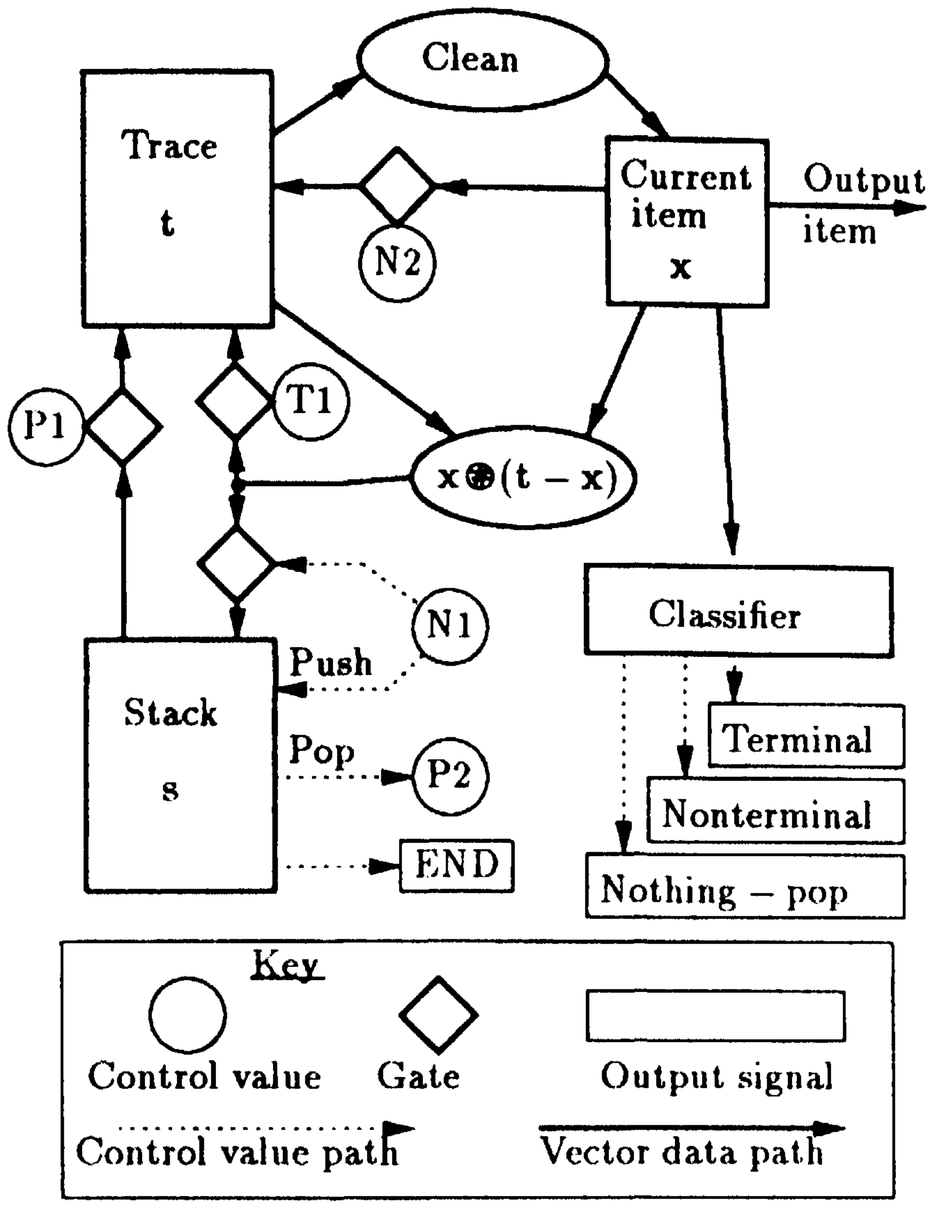

Figure 5: A chunked sequence readout machine.

9 Discussion

Circular convolution is a bilinear operation, and one consequence of the linearity is low storage efficiency. However, the storage efficiency is high enough to he usable and scales linearly. Convolution is endowed with several positive features by virtue of its linear properties. One is that it can be computed very quickly using FFTs. Another is that analysis of the capacity, scaling, and generalization properties is straightforward. Another is that there is a possibility that a system using HRRs could retain ambiguity while processing ambiguous input.

Convolution could be used as a fixed mapping in a connectionist network to replace one or more of the usual weight-matrix by vector mappings. Activations could be propagated forward very quickly using FFTs, and gradients could be propagated backward very quickly using FFTs as well. Such a network could learn to take advantage of the convolution mapping and could learn distributed representations for its inputs.

Memory models using circular convolution provide a way of representing compositional structure in distributed representations. The operations involved are linear and the properties of the scheme are relatively easy to analyze. There is no learning involved and the scheme works with a wide range of vectors. Systems employing this representation need to have an error-correcting auto-associative memory.

Acknowledgements

Conversations with Jordan Pollack, Janet Metcalfe, and Geoff Ilinton have been essential to the development of the ideas expressed in this paper. This research has been supported in part by the Canadian Natural Sciences and Engineering Research Council.

References

[AIJ, 1990] Sp ecial issue on connectionist symbol processing. *Artificial Intelligence,* 46(1-2), 1990.

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

[Hinton, 198l] G. E. Hinton. Implementing semantic networks in parallel hardware. In *Parallel Models of Associative Memory.* Hillsdale, NJ: Erlbaum, 1981.

[Hinton, 1984] G. E. Hinton. Distributed representations. Technical Report CMU-CS-84-157, Carnegie-Mellon University, Pittsburgh PA, 1984.

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

[Hopfield, 1982] J.J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences U.S.A.,* Y9:2554-2558, 1982.

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

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

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

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

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

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

[Plate, 1991] T. A. Plate. Holographic reduced representations: Convolution algebra for compositional distributed representations. Technical Report. CRG-TR-91-1, University of Toronto, 1991.

[Schonemann, 1987] P. H. Schonemann. Some algebraic relations between involutions, convolutions, and correlations, with applications to holographic memories. *Biological Cyberne ics,* 56:367-374, 1987.

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

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

[Touretzky and Hinton, 1985] D. S. Touretzky and G. E. Hinton. Symbols among the neurons: Details of a connectionist inference architecture. In *IJCAI 9,* pages 238-243, 1985.

[Willshaw, 1981] D. Willshaw. Holography, associative memory, and inductive generalization. In *Parallel models of associative memory.* Erlbaum, Hillsdale, NJ, 1981.