A PARSING ARCHITECTURE BASED ON DISTRIBUTED MEMORY MACHINES

Jon M. Slack
Department of Psychology
Open University
Milton Keynes MK7 6AA
ENGLAND

ABSTRACT

The paper begins by defining a class of distributed memory machines which have useful properties as retrieval and filtering devices. These memory mechanisms store large numbers of associations on a single composite vector. They provide a natural format for encoding the syntactic and semantic constraints associated with linguistic elements. A computational architecture for parsing natural language is proposed which utilises the retrieval and associative features of these devices. The parsing mechanism is based on the principles of Lexical Functional Grammar and the paper demonstrates how these principles can be derived from the properties of the memory mechanisms.

I INTRODUCTION

Recently, interest has focussed on computational architectures employing massively parallel processing [1,2]. Some of these systems have used a distributed form of knowledge representation [3]. This type of representation encodes an item of knowledge in terms of the relationships among a collection of elementary processing units, and such assemblages can encode large numbers of items. Representational similarity and the ability to generalize are the principal features of such memory systems. The next section defines a distributed memory machine which incorporates some of the computational advantages of distributed representations within a traditional von Neumann architecture. The rest of the paper explores the properties of such machines as the basis for natural language parsing.

II DISTRIBUTED MEMORY MACHINES

Distributed memory machines (DMM) can be represented formally by the septuple $DMM=(V,X,Y,Q,q_0,\beta,\lambda)$, where

V is a finite set denoting the total vocabulary;
X is a finite set of inputs, and $X \subset V$;
Y is a finite set of acceptable outputs and $Y \subset V$;
Q is a set of internal states;
q_0 is a distinguished initial state;
$\beta: QxX \rightarrow Q$, the retrieval function;
$\lambda: Q \rightarrow QxY$, the output function.

Further, where Y′ denotes the set of all finite concatenations of the elements of the set Y, $Q \subset Y'$, and therefore $Q \subset V'$. This statement represents the notion that internal states of DMMs can encode multiple outputs or hypotheses. The vocabulary, V, can be represented by the space I^k, where I is some interval range defined within a chosen number system, N; $I \subset N$. The elements of X, Y and Q are encoded as k-element vectors, referred to as **memory vectors**.

A. Holographic Associative Memory

One form of DMM is the holographic associative memory [4,5,6] which encodes large numbers of associations on a single composite vector. Items of information are encoded as k-element zero-centred vectors over an interval such as [-1,+1]; $\langle X\rangle=(\ldots x_{-1},x_0,x_{+1},\ldots)$. Two items, <A> and <B> (angular brackets denote memory vectors), are associated in memory through the operation of **convolution**. This method of association formation is fundamental to the concept of holographic memory and the resulting associative trace is denoted <A>*<B>. The operation of convolution is define by the equation $(\langle A\rangle*\langle B\rangle)_m=\sum_i A_i B_{m-i}$ and has the following properties [7]:

Commutative: $\langle A\rangle*\langle B\rangle = \langle B\rangle*\langle A\rangle$,
Associative: $\langle A\rangle*(\langle B\rangle*\langle C\rangle) = (\langle A\rangle*\langle B\rangle)*\langle C\rangle$.

Further, where a delta vector, denoted δ, is defined as a vector that has values of zero on all features except the central feature, which has a value of one, then $\langle A\rangle*\delta = \langle A\rangle$. Moreover, $\langle A\rangle*0 = 0$, where 0 is a zero vector in which all feature values are zero. Convolving an item with an attenuated delta vector (i.e., a vector with values of zero on all features except the central one, which has a value between 0 and 1) produces the original item with a strength that is equal to the value of the central feature of the attenuated delta vector.

The initial state, q_0, encodes all the associations stored in the machine. In this model, associative traces are concatenated (+) through the operations of vector addition and normalization to produce a single vector. Overlapping associative items produce composite

vectors which represent both the range of items stored and the central tendency of the those items. This form of prototype generation is a basic property of distributed memories.

The retrieval function, β, is simulated by the operation of **correlation**. If the state, q_i, encodes the association <A>*<B>, then presenting say <A> as an input, or retrieval key, produces a new state, q_{i+1}, which encodes the item <B>', a noisy version of <B>, under the operation of correlation. This operation is defined by the equation $(<A>\#<B>)_m = \sum_i A_i B_{m+i}$ and has the following properties: An item correlated with itself, autocorrelation, produces an approximation to a delta vector. If two similar memory vectors are correlated, the central feature of the resulting vector will be equal to their similarity, or dot product, producing an attenuated delta vector. If the two items are completely independent, correlation produces a zero vector.

The relation between convolution and correlation is given by

<A>#(<A>*<B>) = (<A>#<A>)*<B> + (<A>#<B>)*<A> + noise ...(1)

where the noise component results from some of the less significant cross products. Assuming that <A> and <B> are unrelated, Equation (1) becomes:

<A>#(<A>*<B>) = δ*<B> + 0*<A> + noise
= <B> + 0 + noise

Extending these results to a composite trace, suppose that q encodes two associated pairs of four unrelated items forming the vector (<A>*<B> + <C>*<D>). When <A> is given as the retrieval cue, the reconstruction can be characterized as follows:

<A>#(<A>*<B> + <C>*<D>)
= (<A>#<A>)*<B> + (<A>#<B>)*<A> + noise
+ (<A>#<C>)*<D> + (<A>#<D>)*<C> + noise
= δ*<B>+0*<A>+noise+0*<D>+0*<C>+noise
= <B> + noise + noise

When the additional unrelated items are added to the memory trace their affect on retrieval is to add noise to the reconstructed item <B>, which was associated with the retrieval cue. In a situation in which the encoded items are related to each other, the composite trace causes all of the related items to contribute to the reconstructed pattern, in addition to producing noise. The amount of noise added to a retrieved item is a function of both the amount of information held on the composite memory vector and the size of the vector.

III BUILDING NATURAL LANGUAGE PARSERS

A. Case-Frame Parsing

The computational properties of distributed memory machines (DMM) make them natural mechanisms for case-frame parsing. Consider a DMM which encodes case-frame structures of the following form:

<Pred>*((<C1>*<P1> + <C2>*<P2> + ...+ <Cn>*<Pn>)

where <Pred> is the vector representing the predicate associated with the verb of an input clause; <C1> to <Cn> are the case vectors such as <agent>, <instrument>, etc., and <P1> to <Pn> are vectors representing prototype concepts which can fill the associated cases. These structures can be made more complex by including tagging vectors which indicate such features as **obligatory case**, as shown in the case-frame vector for the predicate BREAK:

(<agent>*<aniobj+natforce> + <object>*<physobj>
<oblig> + <instrument><physobj>)

In this example, the object case has a prototype covering the category of physical objects, and is tagged as obligatory.

The initial state of the DMM, q_0, encodes the concatenation of the set of case-frame vectors stored by the parser. The system receives two types of inputs, noun concept vectors representing noun phrases, and predicate vectors representing the verb components. If the system is in state q_0 only a predicate vector input produces a significant new state representing the case-frame structure associated with it. Once in this state, noun vector inputs identify the case slots they can potentially fill as illustrated in the following example:
In parsing the sentence **Fred broke the window with a stone**, the input vector encoding **broke** will retrieve the case-frame structure for **break** given above. The input of <Fred> now gives

<Fred>#(<agent>*<Pa>+<obj>*<Po>+<instr>*<Pi>) =
<Fred>#<agent>*<Pa>+<Fred>#<Pa>*<agent> + ... =
0*<Pa>+e_A*<agent> + 0*<Po>+e_B*<obj> +
0*<Pi>+e_C*<instr> =
e_A<agent> + e_B<obj> + e_C<instr>

where e_j is a measure of the similarity between the vectors, and underlying concepts, <Fred> and the case prototype <P_j>. In this example, <Fred> would be identified as the agent because e_B and e_C would be low relative to e_A. The vector is "cleaned-up" by a threshold function which is a component of the output function, λ. This process is repeated for the other noun concepts in the sentence, linking <window> and <stone> with the object and instrument cases, respectively. However, the parser requires additional machinery to handle the large set of sentences in which the case assignment is ambiguous using semantic knowledge alone.

B. Encoding Syntactic Knowledge

Unambiguous case assignment can only be achieved through the integration of syntactic and semantic processing. Moreover, an adequate parser should generate an encoding of the grammatical relations between sentential elements in addition to a semantic representation. The rest of the paper demonstrates how the properties of DMMs can be combined with the ideas embodied in the theory of **Lexical-functional Grammar** (LFG) [8] in a parser which builds both types of relational structure.

In LFG the mapping between grammatical and semantic relations is represented directly in the semantic form of the lexical entries for verbs. For example, the lexical entry for the verb **hands** is given by

hands: V, (↑participle) = NONE
(↑tense) = PRESENT
(↑subj num) = SG
(↑pred) = HAND[(↑subj)(↑obj2)(↑obj)]

where the arguments of the predicate HAND are ordered such that they map directly onto the arguments of the semantic predicate-argument structure. The order and value of the arguments in a lexical entry are transformed by lexical rules, such as the passive, to produce new lexical entries, e.g., HAND[(↑byobj)(↑subj)(↑toobj)]. The direct mapping between lexical predicates and case-frame structures is encoded on the case-frame DMM by augmenting the vectors as follows:

Hands:- <HAND>*(<agent>*<Pa>*<subj> +
<object>*<Po>*<obj2>+<goal>*<Pg>*<obj>)

When the SUBJ component has been identified through syntactic processing the resulting association vector, for example <subj>*<John> for the sentence **John handed Mary the book**, will retrieve <agent> on input to the CF-DMM, according to the principles specified above. The multiple lexical entries produced by lexical rules have corresponding multiple case-frame vectors which are tagged by the appropriate grammatical vector. The CF-DMM encodes multiple case-frame entries for verbs, and the grammatical vector tags, such as <PASSIVE>, generated by the syntactic component, are input to the CF-DMM to retrieve the appropriate case-frame for the verb.

The grammatical relations between the sentential elements are represented in the form of **functional structure** (f-structures) as in LFG. These structures correspond to embedded lists of attribute-value pairs, and because of the **Uniqueness** criterion which governs their format they are efficiently encoded as memory vectors. As an example, the grammatical relations for the sentence **John handed Mary a book** are encoded in the f-structure below:

$$\begin{bmatrix} \text{SUBJ} & \begin{bmatrix} \text{NUM} & \text{SG} \\ \text{PRED} & \text{'JOHN'} \end{bmatrix} \\ \text{TENSE} & \text{PAST} \\ \text{PRED} & \text{'HAND[(SUBJ)(OBJ2)(OBJ)]'} \\ \text{OBJ} & \begin{bmatrix} \text{NUM} & \text{SG} \\ \text{PRED} & \text{'MARY'} \end{bmatrix} \\ \text{OBJ2} & \begin{bmatrix} \text{SPEC} & \text{A} \\ \text{NUM} & \text{SG} \\ \text{PRED} & \text{'BOOK'} \end{bmatrix} \end{bmatrix}$$

The lists of grammatical functions and features are encoded as single vectors under the + operator, and the embedded structure is preserved by the associative operator, *. The f-structure is encoded by the vector

(<SUBJ>*(<NUM>*<SG>+<PRED>*<JOHN>) + <TENSE>
<PAST> + <PRED>(<HAND>*(<↑SUBJ>*<↑OBJ2>*
<↑OBJ>)) + <OBJ>*(<NUM>*<SG>+<PRED>*<MARY>)+
<OBJ2>*(<SPEC>*<A>+<NUM>*<SG>+<PRED>*<BOOK>))

This compatibility between f-structures and memory vectors is the basis for an efficient procedure for deriving f-structures from input strings. In LFG f-structures are generated in three steps. First, a context-free grammar (CFG) is used to derive an input string's constituent structure (C-structure). The grammar is augmented so that it generates a phrase structure tree which includes statements about the properties of the string's f-structure. In the next step, this structure is condensed to derive a series of equations, called the functional description of the string. Finally, the f-structure is derived from the f-description. The properties of DMMs enable a simple procedure to be written which derives f-structures from augmented phrase structure trees, obviating the need for an f-description. Consider the tree in figure 1 generated for our example sentence:

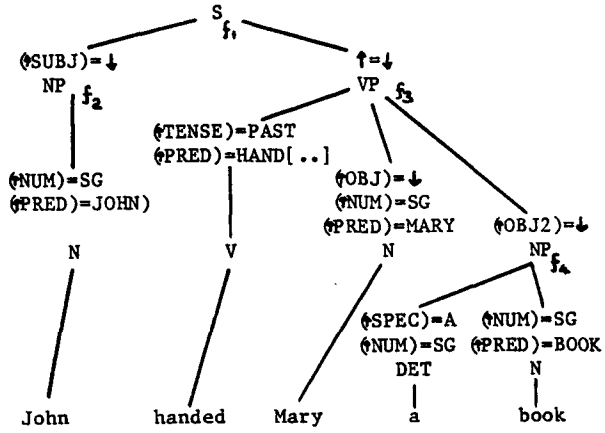

Figure 1. Augmented Phrase Structure Tree

The f-structure, encoded as a memory vector, can be derived from this tree by the following procedure. First, all the grammatical functions, features and semantic forms must be encoded as vectors. The ↓-variables, f_1-f_4, have no values at this point; they are derived by the procedure. All the vectors dominated by a node are concatenated to produce a single vector at that node. The symbol '=' is interpreted as the association operator ,*. Applying this interpretation to the tree from the bottom up produces a memory vector for the value of f_1 which encodes the f-structure for the string, as given above. Accordingly, f_2 takes the value (<↑NUM>*<SG>+<↑PRED>*<JOHN>); applying the rule specified at the node, (f_1 SUBJ)=f_2 gives <↑SUBJ>*(<↑NUM>*<SG>+<↑PRED>*<JOHN>) as a component of f_1. The other components of f_1 are derived in the same way. The front-end CFG can be veiwed as generating the control structure for the derivation of a memory vector which represents the input string's f-structure.

The properties of memory vectors also enable the procedure to automatically determine the consistency of the structure. For example, in deriving the value of $f_{\uparrow}$ the concatenation operator merges the (↑NUM)=SG features for **A** and **book** to form a single component of the $f_{\uparrow}$ vector, (<SPEC>*<A>+<NUM>*<SG>+<PRED>*<MARY>). However, if the two features had not matched, producing the vector component <NUM>*(<SG>+<PL>) for example, the vectors encoding the incompatible feature values are set such that their concatenation produces a special control vector which signals the mismatch.

C. A Parsing Architecture

The ideas outlined above are combined in the design of a tentative parsing architecture shown in figure 2. The diamonds denote DMMs, and the

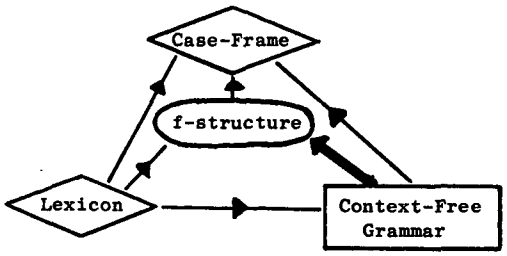

Figure 2. Parsing Architecture

ellipse denotes a form of DMM functioning as a working memory for encoding temporary f-structures.

As elements of the input string enter the lexicon their associated entries are retrieved. The syntactic category of the element is passed onto the CFG, and the lexical schemata {e.g., (↑PRED)='JOHN'}, encoded as memory vectors, are passed to the f-structure working memory. The lexical entry associated with the verb is passed to the case-frame memory to retrieve the appropriate set of structures. The partial results of the CFG control the formation of memory vectors in the f-structure memory, as indicated by the broad arrow. The CFG also generates grammatical vectors as inputs for case-frame memory to select the appropriate structure from the multiple encodings associated with each verb. The partial f-structure encoding can then be used as input to the case-frame memory to assign the semantic forms of grammatical functions to case slots. When the end of the string is reached both the case-frame instantiation and the f-structure should be complete.

IV CONCLUSIONS

This paper attempts to demonstrate the value of distributed memory machines as components of a parsing system which generates both semantic and grammatical relational structures. The ideas presented are similar to those being developed within the connectionist paradigm [1]. Small, and his colleagues [9], have proposed a parsing model based directly on connectionist principles. The computational architecture consists of a large number of appropriately connected computing units communicating through weighted levels of excitation and inhibition. The ideas presented here differ from those embodied in the connectionist parser in that they emphasise distributed information storage and retrieval, rather than distributed parallel processing. Retrieval and filtering are achieved through simple computable functions operating on k-element arrays, in contrast to the complex interactions of the independent units in connectionist models. In figure 2, although the network of machines requires heterarchical control, the architecture can be considered to be at the lower end of the family of parallel processing machines [10].

V REFERENCES

[1] Feldman, J.A. and Ballard, D.H. Connectionist models and their properties. Cognitive Science, 1982, 6, 205-254.

[2] Hinton, G.E. and Anderson, J.A. (Eds) Parallel Models of Associative Memory. Hillsdale, NJ: Lawrence Erlbaum Associates, 1981.

[3] Hinton, G.E. Shape representation in parallel systems. In Proceedings of the Seventh International Joint Conference on Artificial Intelligence, Vol. 2, Vancouver BC, Canada, August, 1981.

[4] Longuet-Higgins, H.C., Willshaw, D.J., and Bunemann, O.P. Theories of associative recall. Quarterly Reviews of Biophysics, 1970, 3, 223-244.

[5] Murdock, B.B. A theory for the storage and retrieval of item and associative information. Psychological Review, 1982, 89, 609-627.

[6] Kohonen, T. Associative memory - A system-theoretical approach. Berlin: Springer-Verlag, 1977.

[7] Borsellino, A., and Poggio, T. Convolution and Correlation algebras. Kybernetik, 1973, 13, 113-122.

[8] Kaplan, R., and Bresnan, J. Lexical-Functional Grammar: A formal system for grammatical representation. In J. Bresnan (ed.), The Mental Representation of Grammatical Relations. Cambridge, Mass.:MIT Press, 1982.

[9] Small, S.L., Cottrell, G.W., and Shastri, L. Toward connectionist parsing. In Proceedings of the National Conference on Artificial Intelligence, Pittsburgh, Pennsylvania, 1982.

[10] Fahlman, S.E., Hinton, G.E., and Sejnowski, T. Massively parallel architectures for AI: NETL, THISTLE, and BOLTZMANN machines. In Proceedings of the National Conference on Artificial Intelligence, Washington D.C., 1983.