WEIGHT MATRIX = PATTERN OF ACTIVATION

Michael G. Dyer
Margot Flowers
Yih-Jih Alan Wang

April 1988
CSD-880026

Weight Matrix = Pattern of Activation
Encoding Semantic Networks as Distributed Representations in DUAL, a PDP Architecture*

Michael G. Dyer
Margot Flowers
Yih-Jih Alan Wang

AI Lab, CS Department
University of California
Los Angeles, CA 90024

ABSTRACT

Using the insight that WMx = PoA (i.e. that weight matrices and patterns of activation are two sides of the same coin), we have designed and implemented DUAL, a PDP memory management architecture to encode and manipulate a symbolically specified semantic network as a virtual, distributed, directed graph in a PDP architecture. DUAL consists mainly of two PDP networks. The STM network functions as a working memory to temporarily hold all of given node's associations, while the LTM network functions as a long term memory for the entire semantic network. The process of encoding causes DUAL to discover distributed representations for all nodes in the semantic network. As a result, nodes with similar associational links are merged and generalized. The DUAL architecture allows us to manipulate the virtual semantic network at a symbolic level, while at the same time retaining many of the robust features (of learning, generalization, graceful error degradation, etc.) associated with PDP networks.

1. MOTIVATION and APPROACH

Natural language processing (NLP) systems must manipulate very abstract forms of knowledge, e.g. the goals, plans and beliefs of narrative agents. Consider just a fragment of the knowledge needed to understand, and answer questions about, the following narrative segment:

> John promised to watch the baby for Mary. John decided to read a book. The baby crawled outside and fell into the swimming pool.
>
> Q: Why did Mary accuse John of being irresponsible?
> A: John had agreed to protect the baby from harm, but the baby nearly drowned.

The necessary background knowledge (informally stated) includes:

<x promise y that z> implies
    <x communicated to y that x has goal G(x), satisfied by z>
    <y communicated to x that y has goal G'(y), to be agent for x>
<after communications, y believes that G(x) and x believes that G'(y)>
<x watch y for z> implies
    <x attends eyes(x) to y and if possible act A(y) can cause harm to y,
        then x performs act B to block A>
<x can know act A(y) by attending eyes(x) to A(y)>
<babies cannot swim and may drown in water>
<possibility of drowning(x) threatens preservation of health goal>
<irresponsible(x wrt y)> implies <promise by x followed by goal failure for y>

---

* This research is supported in part by a contract with the JTF program of the DoD, monitored by JPL, and by a grant from the ITA Foundation. The simulations were carried out on equipment donated by Hewlett Packard.

The traditional approach to the NLP representation problem is to hand code symbolic structures in a Von Neumann architecture, instantiate instances of such structures and then bind them through symbolic operations. While the behavior on limited texts is impressive, the resulting complexity of knowledge interactions and the fragility of such systems is well known (Dyer 1983). Connectionists hope to overcome these problems through automatic learning of distributed structures.

1.1. Approaches to Encoding Abstract Knowledge in Connectionist Architectures.

Faced with the task of encoding such abstract forms of knowledge, we can take any of several approaches:

1. *Brute Force Training:* E.g., create thousands of sample narratives, in which a character acts irresponsibly. Feed them to your favorite ANS (artificial neural system) and perform your favorite adaptive algorithm, e.g. such as back error propagation (BEP) over a multi-layer PDP network. Hope that a notion of "irresponsibility" will magically emerge. Unfortunately, the search space is much too complex. Even humans can only learn about promises, agency, danger, responsibility etc. after having built up intermediate knowledge structures based on prior stages of learning.

2. *Delay:* Wait for neuroscientists to figure out what the brain is doing past the level of surface sensory filtering. This approach, while advocated by some neuroscientists, is unappealing to those interested in exploring theories of abstract semantics, planning and reasoning. Besides, an exploration of the functional/logical constraints imposed by NLP tasks, coupled with architectures to perform these tasks, could supply the theoretical framework needed for neuroscientists to form reasonable hypotheses concerning semantic processing in the brain. Imagine that, say, an old DEC-10, with its virtual memory, multi-tasking operating system, etc. had been sent back to the pre-computational times of the 1940's. Imagine that neuroscientists attempted to form theories of its function by sampling electrons flowing through, say, its bus while observing its input/output behavior for various tasks. What would have been their chance of forming a reasonable theory? How much more complex the task must be when it is the human brain under study.

3. *Symbolic Neuro-Engineering:* This approach takes the following steps: (a) Specify task-related knowledge structures (KSs) and processes symbolically, using known knowledge engineering techniques in AI, (b) encode these KSs into a distributed form within an architecture composed of ANS modules, and (c) use the adaptive capabilities of each ANS module to remove the initial fragility of the KS originally input. This approach allows one to incorporate the synthetic, task-driven orientation of AI knowledge engineering with the bottom-up, analytic approach that dominates in the neurosciences.

1.2. Observation: WMx = PoA

The symbolic neuro-engineering approach motivates one to find techniques for creating distributed knowledge structures. The approach we will describe in this paper is based on an observation by the third author, namely:

> *An entire, multi-layered PDP network, or any network representable as a weight matrix (WMx), can be transformed into a pattern of activation (PoA) in the input layer of a larger network. Likewise, the PoA output by one network can be used to dynamically construct another network (WMx).*

This fundamental relationship can be exploited in many ways. We describe one example below.

2. TASK: Encoding Semantic Networks in PDP Networks

Semantic networks (SNs) are standard ways of representing abstract knowledge. Let us consider a concrete example for discussion -- the knowledge implicit in the following fragment of text:

> John was thirsty. He got a glass and went to the refrigerator. John drank some milk.

A simplified, partial SN to capture the essence of this knowledge appears in Figure 1. Here, nodes postfixed with a "1" are instances of more general nodes. These instances represent the actual events and goals created as a result of conceptual analysis of the input text. The other nodes represent background knowledge needed to analyze the text. This example is greatly simplified, with portions missing.

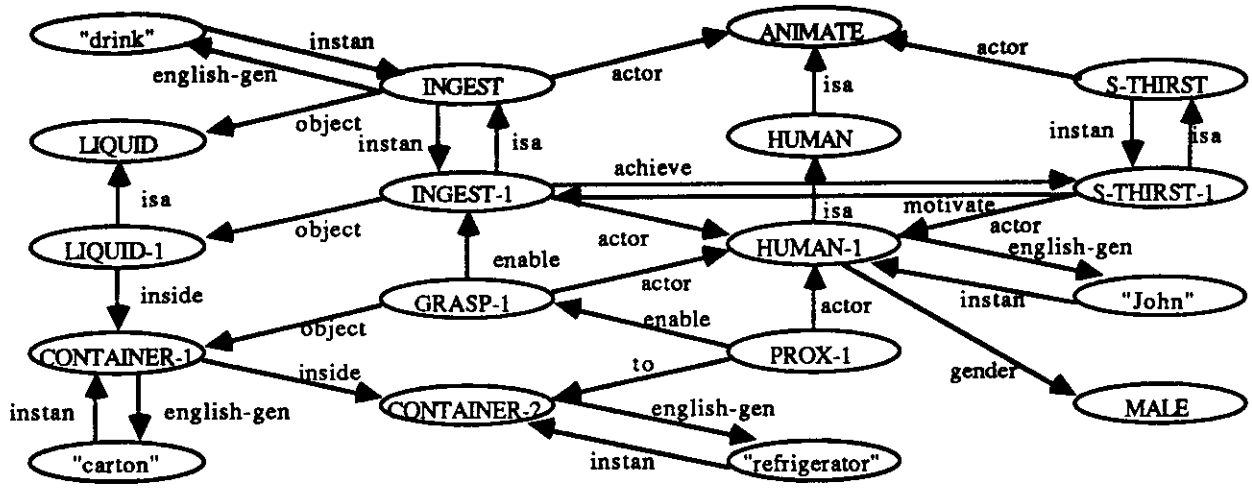
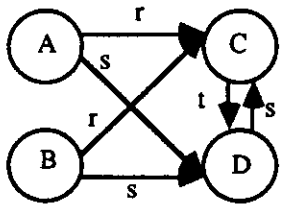


Figure 1. A simplified, partial example of a semantic network.

The careful reader may notice a number of potential problems with it, including: (1) *Back pointers:* An ISA link (from an item to its class) is generally considered unproblematic (e.g. we can easily recall that a dog is an animal) but what about the INSTAN link, (from a class to all of its instances)? Clearly, access to instances should be handled in a more sophisticated manner (e.g., spreading activation or discrimination structures can be incorporated into a SN). (2)*Variables:* Instance nodes encode specific acts or events, but must be linked to general nodes, which encode rules. The encoding of general rules requires variables. For example, the knowledge that INGEST(ACTOR, LIQUID) achieves the goal of S-THIRST(ACTOR) requires that both ACTOR and LIQUID act as variables, and be bound to their instances. This problem can be partially handled by representing named links themselves as nodes. (3) *English Generation:* The English-gen link should be attached only to prototypes (not to instances), but we have done this in the figure for simplicity.

Since we are attempting to show how any SN can be embedded in a PDP network, we will finesse these issues here (which are issues of organization and representation *at the knowledge level*, not the encoding level). For our purposes here, a SN is a directed graph of nodes $\{...A_i...\}$ and named links $\{...r_j...\}$. It can be represented pictorially (Figure 2a) or as a case frame (Figure 2b):



(a)

((A (r C) (s D))
(B (r C) (s D))
(C (t D))
(D (s C)))

(b)

Figure 2: Two ways to represent a SN, (a) graphical representation, and (b) case frame representation.

3. OVERVIEW of DUAL ARCHITECTURE

The essence of our approach is to maintain two PDP networks, STM and LTM (each with input, output, and a hidden layer). STM acts as a short term memory and encodes link-node pairs associated with a given node in the SN. LTM acts as a long term memory and encodes an entire SN. These networks are illustrated in figure 3.

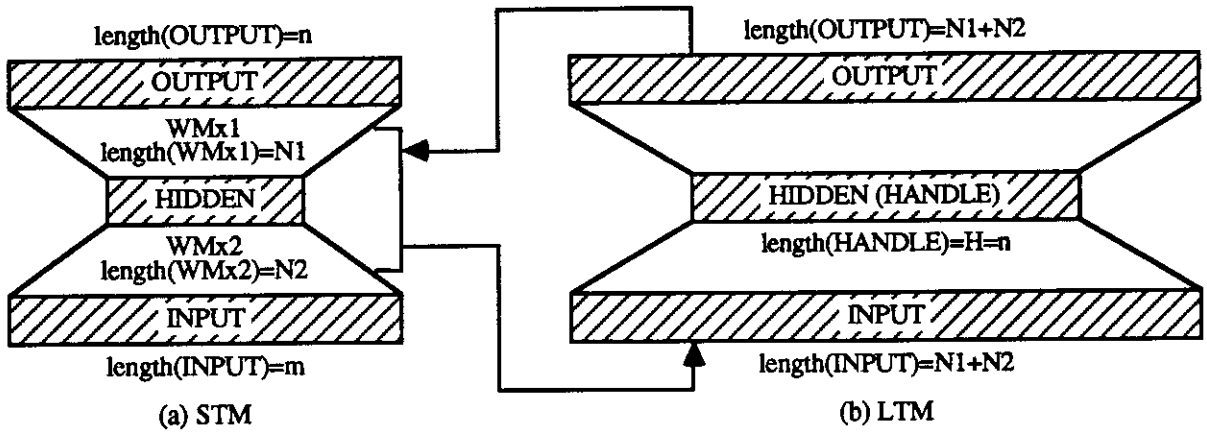


Figure 3. DUAL Architecture, (a) the STM, and (b) the LTM.

STM contains m input and n output units, connected by N1 weights to the hidden units and N2 weights to the output units. S is an entire matrix containing N = N1 + N2 weights. LTM consists of N input units. Thus, the complete weight matrix (WMx) of STM is the input to LTM as a pattern of activation (PoA). LTM acts as an autoassociator. LTM has H hidden units (H=n). Then, a pattern of activation over these hidden units, which we term a *handle*, is used to retrieve an S from LTM. Thus, LTM functions as a method of compressing a number of S's into a single, long term memory weight matrix.

In addition to STM and LTM, DUAL contains a Lexical Memory (LXM). This memory is used to access SN nodes via lexical nodes (e.g. "John", "milk", "drink"). DUAL also contains a memory of handles (HM). In general, LXM associates lexical items with SN nodes. HM holds handles as retrieval cues to LTM. STM holds all (r_j A_j) link-node pairs associated with a given SN node A_i and LTM holds the entire semantic network as a set of autoassociated, STM weight matrices.

4. ENCODING METHOD

Links: Each link (r_j) is represented as an arbitrary, hand-encoded PoA over the input units in STM. In the current version of DUAL, these activity patterns remain unchanged during processing. A set of orthogonal representations is used, in order to minimize interferences, since we want to keep links discriminated.

Nodes: Each node (A_i) is represented by its handle and these PoAs vary dynamically as the SN is encoded. The encoding process involves iterating over all nodes until a consistent set of handles have emerged. In the process of encoding, *DUAL discovers distributed representations for the nodes of the SN*, representations consistent with the desired operations over the SN. To encode an entire symbolic SN (with nodes A_1... A_p and with links r_1... r_m) into LTM, one performs the following four steps (described informally):

1. ENCODE-NODE(A_i)-- to encode one node A_i of SN into STM: Let all of (r_j A_k) link-node pairs associated with A_i be encoded into STM, using standard BEP techniques (Rumelhart and McClelland 1986, chapter 8, pp. 327-329), where the input pattern is r_j and the training pattern (output) is the current handle for A_k. Initially, the representation of each A_k is represented by an arbitrary PoA. The resulting weight matrix will be S_i.

2. SL-TRANSF(S_i) -- transfer S_i to LTM by encoding S_i on LTM's input units and by running BEP over LTM with S_i as the training pattern on its output. Update the resulting (new) handle h in HM.

3. LOAD-LTM -- Iterate ENCODE-NODE and SL-TRANSF for all SN nodes: As each node is encoded in STM, for each link r_j, train STM to respond to the current handle (pointed to by r_j) and then transfer the resulting S to LTM.

4. NORMALIZE -- Iterate steps 1-3 above until the SN has stabilized, i.e. the representations for the nodes (handles) are stable.

The algorithm specifies first iterating over all nodes before updating their representations. In fact, a more dynamic method could be used, whereby nodes are selected at random and their representations upgraded until convergence.

To retrieve the value of link r_i for node A_i (with handle h_i), we do LS-TRANSF(h_i, r_i): (1) Use the handle h_i over the hidden units in LTM and use the associative retrieval properties of LTM to produce an output S_i. (2) Transfer S_i to STM by setting all weights in STM to equal activation values specified in S_i. (3) Clamp the PoA for r_j into the input units of the STM, and a PoA similar to h_k, the handle of the corresponding node A_k, will result in the output units, if (r_j, A_k) is a valid association for A_i.

In a Von Neumann architecture, a unique address can be assigned initially to each node; whereupon the pointer fields for each link can be trivially specified. In DUAL, the representations for each node (A_i) are constantly changing as the representations of the nodes in each link-node pair (associated with A_i) undergo changes. This process occurs because each link must supply a handle for retrieving an entire weight matrix. In DUAL, *each handle points to its own, virtual PDP network*, encoded within another network (i.e., LTM).

5. EXPERIMENTS

To traverse the SN, the operation LS-TRANS is employed. For instance, the questions below can be answered, using the SN in Figure 1:

Q1: Is John a female?
A1: No. John is a male.

Q2: Who was drinking?
A2: John.

One executes the following steps respectively for these two questions:

(1) "John"-->LXM-->handle $h_{"john"}$,
(2) LS-TRANSF($h_{"john"}$, 'instan') -->h_{human1},
(3) LS-TRANSF(h_{human1}, 'gender') -->h_{male}.

(1) "drink"-->LXM-->handle $h_{"drink"}$,
(2) LS-TRANSF($h_{"drink"}$, 'instant')-->h_{ingest},
(3) LS-TRANSF(h_{ingest}, 'instan')-->$h_{ingest1}$,
(4) LS-TRANSF($h_{ingest1}$, 'actor')-->h_{human1},
(5) LS-TRANSF(h_{human1},'english-gen')-->$h_{"john"}$.

We have performed some initial experiments on DUAL. Two are described below. The first one verifies experimentally that the approach described in sections 2-4 works, and briefly examines its retrieval accuracy for various capacities. The second experiment illustrates the nature of the node representations developed during normalization of the SN.

5.1. Accuracy/Capacity Experiment

Once LTM has been constructed, the next step involves retrieving the WMx representation (for a given node and its link-node associations). Due to the nature of PDP networks, a certain degree of inaccuracy is likely to be introduced into these values. Thus, one thing to determine is how the weight matrix representation, recalled from the LTM internal representation, performs for retrieval in STM. The effectiveness of a particular recalled WMx is determined by the accuracy of the output from the STM as compared to the original weight matrix. We have attempted to determine this in three different tests. In each test case, the average number of link-nodes associations for each concept chunk was five. Due to space limitations, we describe only a portion of our results regarding capacity from the first test.

Capacity Test: This test is designed to find the limit of LTM in terms of the number of STM weight matrices (S_i) it can hold while maintaining a satisfactory retrieval accuracy rate. While setting the width of each link to 8 bits, we varied the number of S_i's from 10 to 60, with an interval of 10. In each case, we calculated the average relative error

rate of the original sets of weight matrices right after STM was trained, and of the sets of weight matrices retrieved after LTM was trained. The results are presented in Table below.

Width of Property/Value	Number of Nodes	Relative Error Rate Before LTM Storing	Relative Error Rate After LTM Storing
8	10	1.98%	2.37%
8	20	1.41%	5.96%
8	30	1.79%	22.81%
8	40	2.03%	29.06%
8	50	2.08%	32.98%
8	60	2.39%	30.28%

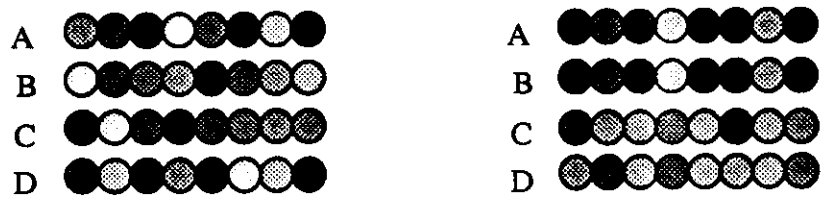
Table 1: Capacity Test Data

Based on our initial tests, we feel that the WMx = PoA approach in DUAL is a promising one. Clearly, more extensive experiments are needed, including comparisons to human memory performance.

5.2. Experiment in Developing Distributed SN Node Representations

The results of this experiment show that *DUAL discovers distributed representations of the SN nodes* input to the system and that these representations are formed to facility the demanded retrieval tasks (i.e. traversing links in the SN).

Similar Nodes are those with similar link-node associations. Before normalization, similar nodes may be represented by very different handles although they are computing similar link-node associations. After normalization, (step 4 in Section 4), however, similar nodes will converge to similar handles. For example, nodes A and B in Figure 2 are similar. Figure 4 illustrates the initial and final representations for all the nodes of the SN in Figure 2. Here, similar nodes (A and B) form handles closer to each other than non-similar nodes, say, A and C.



(a) Before Normalization (b) After Normalization

Figure 4. Representation Developed by DUAL for the SN in Figure 2.

6. DISCUSSION: Generalization and Robustness Without Microfeatures

In the connectionist philosophy, every concept should be collectively represented by an activation pattern over all units. Specific high-level meanings should not be associated with each input, output or hidden unit; otherwise every unit will become a representation of a valid concept by itself. This result contradicts our goal of distributivity. However, since we are dealing with high-level symbolic manipulations, we need to retain a means of accessing specific information about various aspects of each concept (that is, features or properties of a concept).

Microfeatures have been suggested to allow PDP systems to represent high-level concepts while still permitting the networks to generalize over their inputs by having their encodings share microfeatures. However, there are three major problems with microfeatures: (1) *Inapplicability* -- Many microfeatures will be inapplicable for various entities. For example, the microfeature say, METAL = aluminum, copper, etc., will not be applicable for representing people. This requires having a huge number of microfeatures specified ahead of time, with most of them indicated as negative or not-applicable. It seems counterintuitive to represent a person or a building in terms of what they are not. (2) *Tokens vs Types* -- It is awkward to distinguish a specific instance from a general concept

using the microfeature approach. For example, the representations for a car, John's car, a convertible, a broken car, and an antique car must all be differentiated in terms of microfeatures. In the case of "John's car", it might seem sufficient to have some "ownership" microfeatures, but what do they look like? Must we establish an ownership microfeature for each of the possible owners, so that "my car" and "John's car" can be distinguished? If so, not only will we soon be short of enough microfeatures to go around but also each of the "owned by John" and "owned by Bill" microfeatures will begin to look more and more like a localist node. If we do not do this, then how can we solve with microfeatures the problem of representing distinct cars owned by different people? We need a representation mechanism that encodes concept specific information and avoids encoding inapplicable features or properties of the concept. In addition, we want to keep the representations of similar concepts similar. (3) *Flatness* -- Vectors of microfeatures are flat, i.e. they lack the ability to represent recursive and constituent structures. Such structures are necessary to capture grammatical regularities and conceptual relationships, as in "John told Mary that Bill thought Mary wanted to be kissed."

DUAL solves these representation problems by using patterns of activation to represent links that function as normal property names in AI systems. In a semantic network, the node pointed to by a link is interpreted as the value of that property. Only relevant properties (and their values) are associated with a given node. While only a small number of the relevant properties need be used per node, the potential number of different properties representable in a PoA can be very large. By using PoAs also to represent nodes, we solve the type-token problem, since a given PoA can represent a unique person, event, plan instance, object, etc. We also solve the problems of constituent and recursive structures since a semantic network with labelled links can represent such structures directly. At the same time, as we have seen in section 5.2., the representations of these unique objects are fully distributed and objects sharing similar links end up converging to similar patterns of activation.

7. RELATED and FUTURE WORK

7.1. Touretzky and Hinton's Models

Hinton and Touretzky (1985) have also addressed the problem of encoding semantic relations in a PDP architecture. In their case, instead of using a dual network approach, they encode the link r_j from A_i to A_k as a triple (A_i r_j A_k). As a result, the entire semantic network can be encoded in a single PDP network.[1]

At this point, it is not clear exactly what the relative advantages and disadvantages of these two approaches might be. The interference effects are different. When (A_i r_j A_k) is input in their model, it interferes immediately with all other triples in memory. In DUAL, an (r_j A_k) pair can be added to STM without interfering with other node associations in LTM. Of course, once SL-TRANSF occurs, the resulting STM WMx will interfere with other weight matrices in LTM. We are currently designing experiments to see what the differences in interference effects may be.

In Touretzky and Hinton's 1985 model, the representations of each node and link have an equivalent epistemological status. That is, each node and link is treated as a symbol and is represented as a vector of microfeatures. We have already discussed the disadvantages of microfeatures. Nor does their model attempt to dynamically alter the input representations of these symbols to discover distributed representations. In DUAL, distributed representations facilitate the symbolic operations being applied on them.

Touretzky and colleges (Touretzky, 1987; Touretzky and Geva 1987) have more recently developed a model, DUCS, that encodes knowledge in terms of frames, with slots and fillers. Frame notations are equivalent to semantic networks with nodes and labelled links. In DUCS, the representation for slots and fillers are fixed ahead of time and are not allowed to vary dynamically.

To have a slot in one frame F1 point to another frame F2, DUCS make use of the notion of a "reduced description". A "reduced description" for F2, in DUCS, is a subset of the units, able to retrieve F2, in the relevant slot in F1. Traversing a 'pointer' from one frame to another is similar in spirit to DUAL, which accesses a set of node associations from a given handle. However, we feel that the notion of WMx=PoA is more general. Instead of using

---

[1] Touretzky and Hinton also have various short term (or working) memories in their system, but these modules are used for other purposes, e.g. to represent bindings or trigger production rules.

a subset of the units for retrieval, DUAL compresses an entire WMx (using PoA=WMx) and then accesses the handle to retrieve and decode that compression from LTM.

7.2. Pollack's Cascaded Networks

Pollack (1987) describes a a different architecture and task domain, but employs an approach similar in spirit to our own. His cascaded PDP networks consist of a single layer of perceptrons (the context network) connected by multiplicative weights to a 3-layer PDP network (the function network). The cascaded model allows one to transfer entire weight matrices between each network. Using his cascaded approach, Pollack shows speed up over BEP (in a single PDP network) for various versions of the parity problem. Pollack also argues, on intuitive grounds, that an approach that allows one to build a weight matrix in one pass, and then use it in another, simplifies the problem space by a version of "divide and conquer". Pollack also shows how a cascaded architecture can learn syntactic grammars for NLP and speculates that more complex NLP problems may require a context network with more than a single layer of connections. For our task, LTM must possess hidden units to allow compression of an entire STM weight matrix into the handle that other nodes in the semantic network can 'point' to.

7.3. Future Work

DUAL is being extended in various directions. First, the current version of DUAL serves as an independent component in a host system which itself works mainly at the symbolic level. We need to systematically replace symbolic processes with PDP components. As stated earlier, more extensive experiments on DUAL's performance are needed, along with comparisons to human memory. One drawback of *all* models that manipulate a single (A_i, r_j, A_k) relationship at a time is that they run sequentially at the knowledge level. This sequentiality is avoided, for example, in marker parsing systems (Charniak, 1986) which operate over all nodes of a SN in parallel. Unfortunately, marker passing systems are symbolic and localist. We are currently exploring ways to increase parallelism for DUAL at the knowledge level.

8. CONCLUSIONS

Using the insight that WMx = PoA (i.e. that weight matrices and patterns of activation are two sides of the same coin), we have designed and implemented DUAL, a PDP memory management architecture to encode and manipulate a symbolically specified semantic network as a virtual, distributed, directed graph in a PDP architecture. DUAL consists mainly of two PDP networks. The STM network functions as a working memory to temporarily hold all of given node's associations, while the LTM network functions as a long term memory for the entire semantic network. The process of encoding causes DUAL to discover distributed representations for all nodes in the semantic network. As a result, nodes with similar associational links are merged and generalized. The DUAL architecture allows us to manipulate the virtual semantic network at a symbolic level, while at the same time retaining many of the robust features (of learning, generalization, graceful error degradation, etc.) associated with PDP networks.

REFERENCES

Charniak, E. A Neat Theory of Marker Passing, *Proc. of the National Conference on Artificial Intelligence* (AAAI-86), Philadelphia, PA, 1986.

Dyer, M. G. *In-Depth Understanding*. Cambridge, MA: MIT Press, 1983.

McClelland, J. L. and Kawamoto, A. H. Mechanisms of Sentence Processing: Assigning Roles to Constituents of Sentences. in *Parallel Distributed Processing*, Vol. 2. MIT Press/Bradford Books, Cambridge, MA, 1986.

Pollack, J. B. Cascaded Back-Propagation on Dynamic Connectionist Networks. *Proc. of Ninth Annual Conference of the Cognitive Science Society*, Seattle Wash. 1987.

Rumelhart, D. E. and McClelland, J. L. Learning Internal Representations by Error Propagation. Rumelhart & McClelland. *Parallel Distributed Processing*. Cambridge, MA: MIT Press/Bradford Books, 1986.

Touretzky, D. S. and Hinton, G. E. Symbols among the neurons: details of a connectionist inference architecture. *Proc. of International Joint Conference on Artificial Intelligence* (IJCAI-85), Los Angeles, CA, 1985.

Touretzky, D. S. Representing Conceptual Structures in a Neural Network. *Proc. of IEEE First International Conference on Neural Networks*, San Diego, CA, 1987.

Touretzky, D. S. and Geva S. A Distributed Connectionist Representation for Concept Structures. *Proc. of the Ninth Annual Conference of the Cognitive Science Society*, Seattle, WA, 1987.