REPORT:

__Mark A. Jones__ is a member of the technical staff in the Digital Systems Research Department of AT&T Bell Laboratories in Murray Hill, New Jersey. He joined AT&T in 1984 and is working on distributed processing models for artificial intelligence tasks such as natural language processing and reasoning. Mr. Jones has a B.S. in electrical engineering from Rice University, an M.A. in education from Stanford University, and an M.S. and a Ph.D. in computer science from the University of Kansas.

PROGRAMMING CONNECTIONIST ARCHITECTURES

Introduction

Connectionist models of computation represent a program as a set of weighted connections between nodes in a directed graph. Each node has an associated local algorithm for combining values from its input connections and passing the result along its output connections. The models are sometimes termed "electronic neural networks" because they are similar to abstract models of neural processing. The models also resemble the type of fine-grained parallelism in data flow architectures. Although still an emerging technology, electronic neural networks have many advantages over standard computational models, including fault tolerance, the ability to make graded judgments, and quick convergence to approximate solutions.

However, finding a successful programming method is one of the great challenges for connectionist processing models. This paper highlights three current programming alternatives:

- Explicit programming of connectivity and weight assignments.
- Learning algorithms that can be used to train a network starting from some (generally random) initial configuration.
- Compiling approaches that take an abstract specification and translate it into a network description.

Each of these approaches has advantages and disadvantages.

Explicit Programming

Explicit programming is perhaps the most frequently used. After the problem has been analyzed, a network is carefully constructed with hand-tailored link weights and node thresholds. The construction may be entirely ad hoc, or the problem may be recast mathematically as an optimization problem in a form that is known to yield networks with desirable convergence properties. For example, the traveling salesman problem can be viewed in terms of energy minimization in a Hopfield network. Here, units represent city-traversal position pairs and link weights represent distances and problem constraints.[1] Once the problem has been represented, there is limited control over the initial starting state of the system and the speed with which it stabilizes. There is no particular method for recasting a novel problem in terms of energy minimization, although there are common techniques such as mutually inhibitory "winner-take-all" configurations that force mutual exclusion.

Learning Algorithms

Most network learning algorithms use a form of supervised learning to adjust the connection strengths and reduce the observed error in the network response. For example, the back-propagation learning algorithm[2] trains a feed-forward network to act as a transducer between input and output.

Such learning approaches can avoid explicit weight setting (although good training sets can be difficult to build), and can develop internal generalizations in a distributed representation that are tolerant to input noise and the failure of particular units. Distributed representations use patterns of activity across numerous units to represent concepts and perform computations among them. Many theoretical questions remain regarding learning time, the ability to create good internal generalizations, and the appropriate network topology (i.e., the number of layers in the network, the number of units in each layer and the connectivity between layers). At Bell Laboratories in Holmdel, New Jersey, researchers are currently working on many of these questions.

Compiling Approaches

The third method of programming connectionist models is most like the standard compiling approach for traditional computers. One of the greatest obstacles to this alternative is the lack of a general computational model for the popular connectionist architectures (whether analog or digital, deterministic or stochastic). It has been difficult to use traditional notions, such as states of various computations and transitions between them, or symbolically representing and manipulating concepts.

Before we can approach moderately complex tasks, such as parsing recursive languages, we need solutions to these problems. Past approaches have ranged from expanding the recursive grammar to the degree necessary for inputs of a particular fixed length[3] to using a hybrid approach combining connectionist subsystems with more traditional processing models.[4]

Active Production Networks

We have been developing a computational model for fine-grained parallelism, called *Active Production Networks* (APN), with a somewhat different approach than much of connectionism. Rather than starting from a presumed neural model and trying to understand what it can compute, we have started with *rule-based systems*, a well-understood computational framework, to try to understand how to coax fine-grained parallelism from them. APNs are somewhat more powerful than most connectionist models, and deal explicitly with difficult aspects of rule-based systems such as maintaining multiple, simultaneously active instances of concepts and handling coindexing constraints.[5]

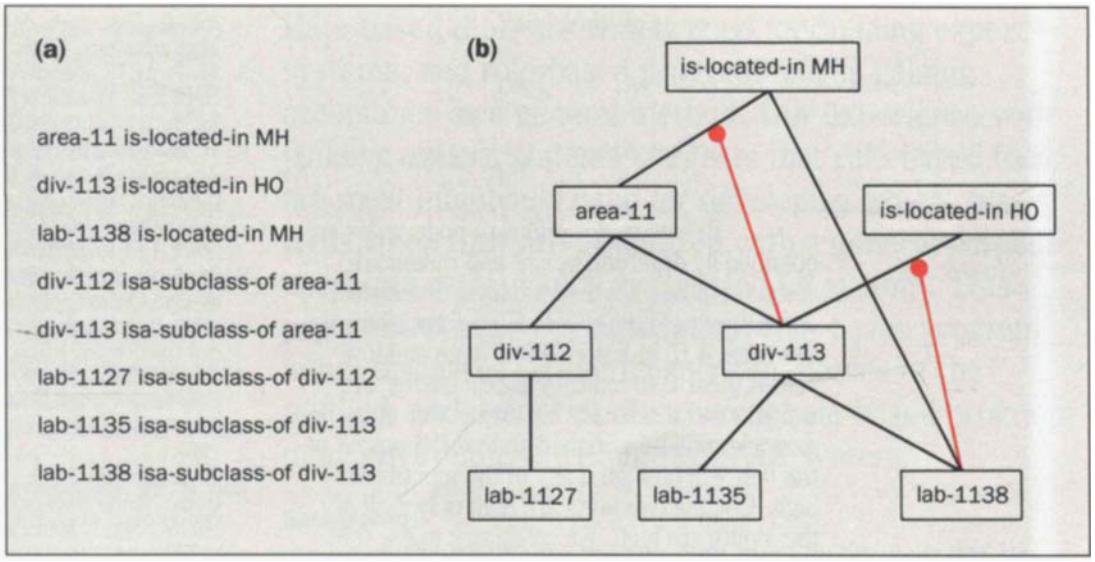
For example, in parsing natural language sentences, there are often many instances of noun phrases or clauses that are recursively embedded. Subject-verb number agreement, a form of coindexing, must be maintained for each present-tense clause, even when they are nested.

The APN model has been applied to several problems in artificial intelligence, including (a) natural language recognition and generation, (b) information retrieval, and (c) the important problem of taxonomic inheritance reasoning that arises in knowledge representation systems, expert systems, and object-oriented programming languages. Inheritance reasoning uses only a subset of the APN model, but is sufficient to illustrate some of the important concepts.

Figure 1(a) shows a set of facts to be handled by an inheritance reasoner. These facts contain information about organizational entities, relationships between them, and their properties. Strictly speaking, we must also add to these facts the knowledge that

- The `is-located-in` property is unique and inheritable across the `isa-subclass-of` relationship.
- `MH` and `HO` are mutually exclusive.

Figure 1. Taxonomic inheritance: (a) A set of inheritance facts; (b) An APN (Active Production Network) inheritance network. Inhibitory links are shown in color.


(a)

area-11 is-located-in MH
div-113 is-located-in HO
lab-1138 is-located-in MH
div-112 isa-subclass-of area-11
div-113 isa-subclass-of area-11
lab-1127 isa-subclass-of div-112
lab-1135 isa-subclass-of div-113
lab-1138 isa-subclass-of div-113

Negation is treated as one type of mutual exclusion. Inheritance descriptions need not be tree structured and can contain multiple inheritance paths that may override one another.

Given these facts, we can compile the network representation shown in Figure 1(b). The network contains disjunctive units (e.g., `area-11`) and inhibitory units (inhibitory links are shown in color) that are inserted by the compiler where the inferences must be overridden. A distributed inheritance computation proceeds by passing values through the network from an object to its properties. The network can deduce inherited facts such as

`lab-1135 is-located-in HO.`

The network also reflects the fact that more specific properties such as

`lab-1138 is-located-in MH`

override more general properties such as

`div-113 is-located-in HO`

which itself is exceptional to the more general

`area-11 is-located-in MH.`

(See Etherington.)[6]

APN inheritance networks that include disjunction, conjunction and inhibition nodes can be translated to run on the type of hardware neural networks described by Howard, Jackel, and Graf in the article on pages 58-64 of this issue.[7]

Neural network units often perform a simple, sigmoidal function of a weighted sum (*net*) of input values and a bias (θ), which functions as a negative threshold:

$$net = \sum w_j i_j$$

$$\sigma[net, \theta] = \frac{1}{1 + e^{-(net + \theta)}}$$

Behaviors for different node types are obtained by different weight and threshold assignment policies that preserve the logical correctness of inferences. High activation values (near 1.0) indicate logical truth and low values (near 0.0) indicate logical falsity. The weights and thresholds for disjunctive units, for example, will be assigned so that the output of the unit will be high if any of the inputs are high. Conjunctive units are high only if all of the inputs are high. For inhibitory units, the output can be pulled down by any of the inhibiting inputs.

An advantage of the uniform, sigmoidal units is the existence of supervised learning algorithms for such networks that incrementally adjust the weight and threshold values.

We are currently looking for ways to combine knowledge from rules with knowledge from training networks. That is, it may be possible to compile what we know (or think we know!) into the network and then further fine-tune, broaden and generalize the performance by training. This could reduce training time for the system, help eliminate statistical artifacts of the training order, and speed up the system on exceptional and difficult cases.

References

1. J. J. Hopfield and D. W. Tank, " 'Neural' Computation of Decisions in Optimization Problems," *Biological Cybernetics*, No. 52, 1985, pp. 141-152.
2. D. E. Rumelhart and J. L. McClelland, *Parallel Distributed Processing*, MIT Press, Cambridge, Massachusetts, 1986.
3. M. Fanty, "Context-Free Parsing in Connectionist Networks," *Technical Report 174*, University of Rochester Computer Science Department, 1986.
4. D. S. Touretzky and G. E. Hinton, "Symbols Among the Neurons: Details of a Connectionist Inference Architecture," *Proceedings of the 9th International Joint Conference on Artificial Intelligence*, Los Angeles, California, 1985, pp. 244-248.
5. M. A. Jones, "Coindexing as a Feedback Mechanism in Connectionist Architectures," *Proceedings of the 10th International Joint Conference on Artificial Intelligence*, Milan, Italy, 1987, pp. 602-610.
6. D. W. Etherington, *Reasoning from Incomplete Information*, Pitman Research Notes in Artificial Intelligence, Pitman Publishing Limited, London, 1987.
7. R. E. Howard, L. D. Jackel, and H. P. Graf, "Electronic Neural Networks," *AT&T Technical Journal*, Vol. 67, No. 1, January/February 1988, pp. 58-64.

(Manuscript received October 30, 1987)