An Experimental Architecture that supports Non-Temporal Prediction

Paul Robertson
University of Texas at Dallas
Natural Sciences & Mathematics
Box 688, Richardson
Texas 75080, USA

Abstract

A constructive theory of memory organisation has been developed, based upon the principle of non-temporal prediction. The theory predicts much of the experimental findings on recall and forgetting and provides a computational foundation for some of the intuitive notions of the society of mind theory. This paper describes an experimental architecture that is being used to study this form of learning. The architecture is a highly distributed system that achieves 'structural' learning through the application of a particularly powerful form of natural constraint.

Keywords - Non-Temporal Prediction, Distributed Problem solving, Society of Mind, Models of Learning, Skill Acquisition.

Introduction

Progress in VLSI techniques along with the emergence of some highly distributed architectures such as Fahlman [1] and Hillis [2] has awakened an interest in examining what can be done with certain architectures based on simple 'neuron like' processors, such as Hinton [3 4] and Feldman [5]. A theory of learning based on the principle of non-temporal prediction has been developed that is completely data-driven [6 7]. In this paper, we describe an experimental architecture that is being used to study this form of learning.

Non-Temporal Prediction

Learning and memory can be viewed as mechanisms for the acquisition of knowledge. Knowledge itself can be viewed as a means of predicting events in the world. Our survival is in a large part dependant on our ability to 'predict' the world. It is supposed that learning has evolved to meet this need. Making predictions about the world can be classified into two broad categories. First, there is the class of predictions that are time related. An understanding of 'Gravity' might be classified in this way, to understand 'gravity' is to predict that when a thing is dropped it will fall to the ground (or the class of predictions of which that is a simple example). This form of prediction is time related because the two defining events (the dropping and the hitting on the floor) are disparate in time. The second category, to which this paper is specifically addressed, concerns predictions that are unrelated to time. This kind of prediction concerns the classification of events. Here, learning the concept of an 'arch' (say) is making a prediction about what objects constitute 'arch'. When examples of arches that conform to this prediction are encountered they will be recognised as such, just as dropping an object that subsequently falls to the ground is recognised as indicating the presence of 'gravity'. The difference, is that the second category is unrelated to time. There are several reasons why it is useful to make this form of distinction.

(1) Many theories of learning and forgetting are based upon the notion of trace decay. Recency explains certain observable phenomenon, but is difficult to justify computationally and gives rise to some serious problems when dealing with predictive situations of vastly disparate times.

(2) Many of the effects for which recency was proposed can be adequately explained without reference to 'time' or 'trace decay'.

(3) Many problems that at first appear to be temporal in nature can be expressed in terms of the non-temporal paradigm. It is not known whether all situations can be transformed in this way. It may be that learning for 'temporal' situations is itself a learned strategy, there is some evidence to support this conjecture.

(4) It is possible to solve the problem of non-temporal prediction computationally in terms of a highly distributed architecture of simple processors.

Learning by Modification

The notion that learning usually takes the form of modifying an existing skill is intuitively attractive. Many attempts at capturing this intuition computationally have been tried, STRIPS [8] employed an augmented triangle table that allowed old plans to be 'modified' to suit new situations, an idea recently extended by Carbonell [9]. Minsky [10] discussed a form of learning in which new agents arise by 'splitting off' from old ones, with only small changes and essentially the same data connections. The mechanism presented in this paper follows the spirit of Minsky's 'agent splitting' but differs in detail. The architecture presented differs in that instead of splitting a single process (by copying) and then modifying the copy, it supports multiple copies of (almost) identical agents. Learning involves taking a 'suitable subset' of these agents and modifying it. Before describing the architecture itself, we should make a few points regarding the significance of this difference.

(1) It seems likely that natural systems such as the human brain can support this form of 'redundency'.

(2) Having multiple copies introduces a degree of 'fault tollerence', in particular, the 'Grandmother Problem' does not arise.

(3) Most significantly, having many copies means that a data driven mechanism can be utilized to achieve the 'split' instead of needing a top down decision to split.

Understanding Discontinuous Changes in Capability

Instead of having a single agent that can perform a given task, the architecture supports many such agents. We will refer to a set of similar agents as a process-set. The agents of a process-set compete to influence the state of the system. Each agent provides its own prognosis and some indication of how reliable it believes this prognosis to be (based on a simple probabilistic analysis). One agent's prognosis will be chosen as the most credible alternative. The computation of credibility will also be computed on the basis of a simple probabilistic analysis. Instead of hypothesizing that when a thing is learned its strength gradually increases, or when it is forgotten, it gradually decreases (trace decay), this model of learning distinguishes several phases of learning. First, the agent is generated in isolation (we will demonstrate one algorithm for agent creation when expounding the details of the architecture). Then, the agent must be refined

(discover its own boundaries and be able to accurately compute the reliability of its own prognosis). Finally, the agent must be discovered by other agents already in the system. This final stage is one in which the agents credibility is computed as the result of a probabilistic analysis, and corresponds closely to the notion of forming K-lines expounded by Minsky [11]. When a new and necessary agent is created, its success causes its credibility to rise until enough samples have been obtained to raise its credibility to a level above that of the previous 'favorite' agent for this task. At this point, the new agent will suddenly be used in place of the previous favorite, giving rise to an observable discontinuous change in performance.

The Experimental Architecture

The experimental architecture can be described at several levels. At one level, is the general system topology defined by a number of intuitive connectivity restrictions described in [7] and in more detail in [6]. Space prevents a discussion of this aspect of the architecture. The heirarchy can be decomposed into neighborhoods of agents that will, for the purposes of this paper be totally connected (the overall heirarchy allows the connectivity complexity to be kept linear despite the total connectivity within neighborhoods, furthermore, the connectivity within a neighborhood can be relaxed [2] without loss of generality). A neighborhood contains two computationally distinct components. The processors, that may be programmed to compute a predictive rule, and the creators that program processors for the purposes of generating new agents (learning) and replenishing process-set size when process splitting has resulted in an insufficient process-set cardinality (housekeeping). We will discuss the creator and the processor objects seperately. A programmed processor will be refered to as an agent.

The Anatomy of a Neighborhood

Consider a neighborhood to be a two dimensional sheet of processing elements. Each processor in the region recieves an input from outside the neighborhood, being totally connected each processor also recieves inputs from the outputs of every other processor in the neighborhood.

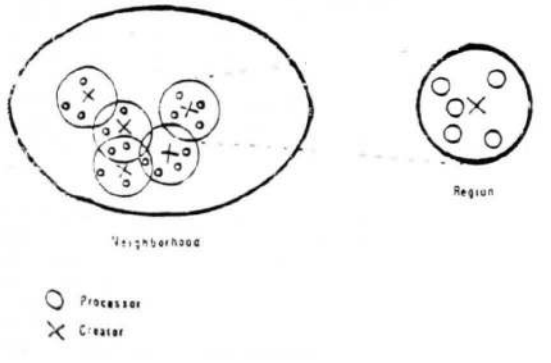

Figure 1

The neighborhood itself is divided into smaller overlaping 'regions' (See Figure 1). Each Region contains a single creator and a large number of processors. The creator has access to all local inputs to the processors within the region, and the outputs of each processor in the region. The creator can cause one or more of the processors in its region to be re-programmed.

Computation performed by a Creator

The creator monitors both the inputs local to the region and the number of processors that respond to the input. If too few processors respond to an input, the creator selects the processors that are least successful and re-programs them so as to increase the process-set cardinality. The creator is continually performing the following sequence of computations.

(1) Compute the activity of the inputs to the region. This involves counting the number of active inputs locally. Let the activity be denoted by activity.

(2) Compute the response size. This involves counting the number of processors in the region that responded to the inputs. Let the response size be denoted by response.

(3) Compute the expected response size. In the present system, the expected response size is a linear function of the activity.

(4) If response<expected, re-program response-expected processors. This involves choosing the required number of processors, the least successful ones are chosen first. Each processor keeps a record of its success. In our implementation, each region keeps a sorted list of processors, when n new processors are required, the first n are taken from this sorted list. In a truely parallel system such as might be found in Biological systems, this process can be achieved simply by broadcasting a re-program command to all processors and using a system of inhibition to prevent re-programming of the better processors (for a development of this idea see [6]).

Processing Inputs

It is convenient to describe the operation of the processors in two stages. First, how each input to a processor is handled on an individual basis, and second how these inputs are combined to form a prognosis.

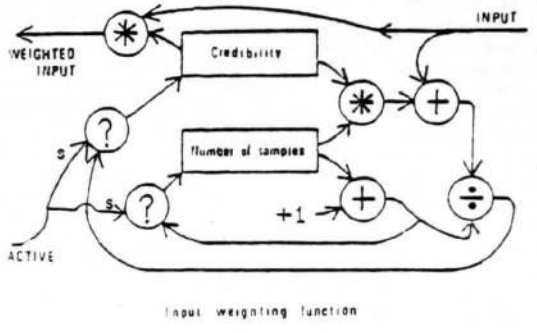

Figure 2

Each input to a processor is processed by an input weighting function. Figure 2 illustrates the function of this process. Each input weighting function (corresponding to $input_i$) samples its input whenever the process is active. In this way, the input weighting function computes for its input, the credibility that that input is indicative of the event being diagnosed -- the probability that the input will be active when the event is diagnosed $P(input_i \mid this.agent.active)$.

Other Processor functions.

Once the inputs have been weighted according to their credibility, they can be combined to form the prognosis.

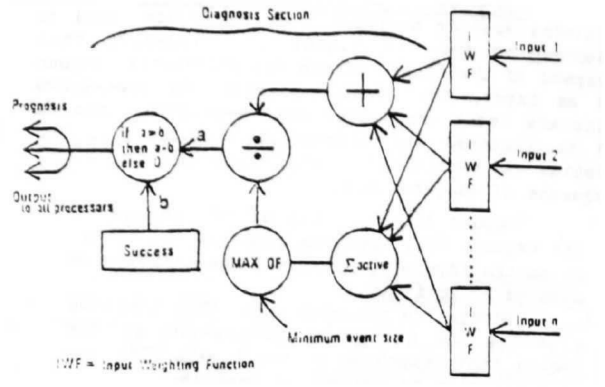

Figure 3

Figure 3 illustrates the basic operation of the diagnosis part of a processor. The value of success is adjusted whenever the agent is active. Space prevents further development of this idea here, however, low success values indicate failure in implementing a predictive rule, and such processors will be re-programmed by their creator when a new agent is required. The inherent limitations of simple Linear Threshold devices such as the prognosis function, are used as a powerful natural constraint. This guarentees that most agents that are created will eventually die (success will fall until it is eventually re-programmed). This gives rise to a very ecenomical use of processors without the need for a knowledge driven resource (processor) allocation system (these ideas are developed in detail in [6]).

Conclusion

Due to a lack of space, many significant details and much of the theory had to be omitted. Experiments with a LISP based implementation of the system outlined in this paper have been encouraging. Complex structural descriptions can been learned by the system. The system is robust in that usually, no agent is so important that its removal will be critical (due to duplication), and a high degree of noise can be tollerated. An analysis of the systems noise immunity can be found in [6]. It is interesting that as the regions approach saturation (most processors are successfully programmed as agents), it becomes increasingly difficult to learn a new rule. This is because, before a new agent can achieve a respectable success it is re-programmed by its creator because it is still the least successful agent. Only intensive training will result in the new agent being learned, and this will be at the cost of one of the other successful agents. Full details of the architecture, and justification of its design can be found in [6].

References

1. Fahlman,S.E.
NETL: A System for Representing and Using Real-World Knowledge The M.I.T. Press. Cambridge Massachusetts 1979. ISBN 0-262-0609-8

2. Hillis,W.D.
The Connection Machine
M.I.T. AI Memo 646 September 1981.

3. Hinton,G.
A Parallel Computation That Assigns Canonical Object-Based Frames of Reference
Proceedings of IJCAI-7 1981.

4. Hinton,G.F. & Anderson,J.A. (eds)
Parallel models of associative memory.
Hillsdale, NJ: Erlbaum, 1981.

5. Feldman,J.A.
A Connectionist Model of Visual Memory
In [4] above.

6. Robertson,P.
Process Dependant Localized Memory
University of Texas at Dallas Technical Report.

7. Robertson,P.
Non-Temporal Prediction: A Distributed System For Concept Acquisition
Proceedings of the Fourth National Conference of the CSCSI/SCEIO 1982

8. Fikes,R.E. & Nilsson,N.J.
STRIPS: A new Approach to the Application of Theorem Proving to Problem Solving
Artificial Intelligence Journal, Vol. 2, no.3/4, 1971

9. Carbonell,J.G
A Computational Model of Analogical Problem Solving
Proceedings of IJCAI-7. 1981.

10. Minsky,M.
Plain talk about Neurodevelopmental Epistemology
Proceedings of IJCAI-5. 1977.

11. Minsky,M.
K-Lines: A Theory of Memory
In 'Perspectives on Cognitive Science' Donald A. Norman ed.