LEARNING STRATEGIES FOR DYNAMIC DECISION PROBLEMS USING ARTIFICIAL NEURAL NETWORKS[1]

Pankaj Mehra and Benjamin Wah
Coordinated Science Lab.
University of Illinois, MC 228
Urbana, IL 61801, USA

We present the architecture of an AI system for learning strategies in complex domains. SMALL (Strategy Acquisition by MetA-Level Learning) is a system architecture that embodies the principles of modular knowledge-level design and phased training in order to learn strategies in a flexible yet efficient manner. A class of difficult decision problems is identified. It is shown that specific bodies of knowledge can be used to counter the specific aspects of difficulty. Our approach is illustrated by a connectionist implementation of the knowledge modules. This approach can be used for learning load-balancing strategies in loosely coupled multiprocessors.

Dynamic Decision Problems

We have identified a rich class of problems that involve learning to search in real time. These are called dynamic decision problems (hereafter, DDPs). Formally, these are decision problems of the *satisficing* type, with an ill-defined objective function. The objective function, as well as the strategy to achieve it, must be acquired while searching. A DDP is defined by a set of observable parameters X, a set of actions A, and a reinforcement signal r. Under the assumption that a high value of r is indicative of near-optimality, and *vice versa*, the searcher first constructs an internal estimate of an objective function F. This function is such that it attains its maximum value when the reinforcement is highest. Both F and r are functions of a history of (event,action) pairs, where an event is a vector of values for each of the variables in X. Once the objective function is acquired, a variety of techniques are available for learning strategies for that particular objective function.

Load Balancing: an illustrative problem

Load balancing is important in distributed computing systems for distributing workload from heavily loaded processors to lightly loaded processors. Its goal is to improve the utilization of idle resources with a view to getting improved performance through reduced response time. An essential operation in each processor is to find a target processor for sending the jobs. A **load balancing strategy** consists of a *decision policy* and a set of *decision metrics*[2]. The policy specifies how task migration decisions can be made in terms of some higher level parameters of decision, the decision metrics. The problem can be viewed as a *parallel search* for an optimal assignment of resources to tasks, wherein the decisions to migrate tasks are made locally at each of the sites involved.

There are two major tasks for an autonomous, intelligent load balancing system: the design of decision metrics by composition of primitive inputs (*Workload Characterization*), and the learning of decision rules constituting the decision policy for migration (*Workload Distribution*).

Characteristics of Dynamic Decision Problems.

We now examine how the characteristics of the problem class affect the complexity of DDPs. Several characteristic features of problems are considered: the specification of F (the objective function), and, the cardinality of X (the input vector). We also study the effect of the problem-solving environment (the nature of X and F) and the learning environment (the nature of r) on the architecture of the learning system.

(a) Ill-specified objective function.

Consider the response time objective function in load balancing. Even though the function is *measurable* locally at each site, it is not *defined* in terms of the global input parameters. Therefore, the first step in optimization is to *learn* the relationship between the objective function and the locally available inputs *relative to* the performance of the same input on an idle machine. (This step ensures consistency of comparisions between these values.) Assuming that the global objective function is representable and learnable as a function of local objectives, each processor broadcasts a vector of values that reflect its internal states for various types of inputs. Each processor uses its internal local objective value, and the values broadcast by other machines, in order to learn the relationship between X and r, implicitly creating a model of F. (Each processor attempts to optimize the *integral* of r over time. Thus, r is treated like the *slope* of F.)

In load-balancing, this step is called workload characterization. The data on response time becomes available only as pairs $(x, F(x))$. These values must be assimilated into the model of F *incrementally*. Since knowledge of $F(X)$ is fundamental to any optimization strategy, the architecture must have a learning component that learns F dynamically.

(b) Large number of time-varying inputs.

In load balancing, X comprises of various *task parameters* (CPU time, process size, CPU/IO-intensity), *workload parameters* (number of competing processes, recent CPU occupancy, recent disk transfer rate, free memory), *architecture parameters* (CPU FLOPS, average seek time), and *network parameters* (link bandwidths, network traffic). The distribution of X is not assumed stationary. Thus, learning of $F(x)$ must proceed on-line. Learning of F is complicated by the constraints of dimensionality. The module for learning F must be able to simultaneously consider the effect of a large number of numeric parameters. Internally, it should be able to formulate high level decision metrics by forming suitable combinations of primitive parameters.

[1]This research was supported in part by the National Aeronautical and Space Administration under grant NCC-2-481, and in part by National Science Foundation Grant MIP-88-10584

(c) Asynchronous, delayed feedback.

During the operation of the learning system, the reinforcement signal $r(t)$ at time t summarizes the evaluation of balancing decisions before $t-\Delta t$. It is assumed that $r(t) \propto F(X(t-\Delta t))$, in the sense that high, positive reinforcements indicate near-optimality. Now, $X(t-\Delta t)$ results from decisions preceding $t-\Delta t$. Therefore, these decisions must remain in memory so that the feedback r can be distributed temporally among them. This process is called *temporal credit assignment* (hereafter, TCA). The knowledge required for TCA takes the form of a persistence model. The persistence model is used to maintain the persistence of each observed variable that was changed as a result of a particular action. Typically, the persistence of an effect decays exponentially with time elapsed since the performance of the action causing that effect. This body of knowledge concerns the behavior of X over time. It accounts for *natural attrition* in the persistence of a time-varying parameter. The behavior of X over time is also influenced by recent decisions. This creates a need for a body of knowledge that relates actions and persistences. This body of knowledge is called a causal model. The causal model of the environment is used to relate actions and events.

Neither of these models is available *a priori*. Accumulated knowledge tends to become obsolete because the value distributions of X are not assumed *stationary*.

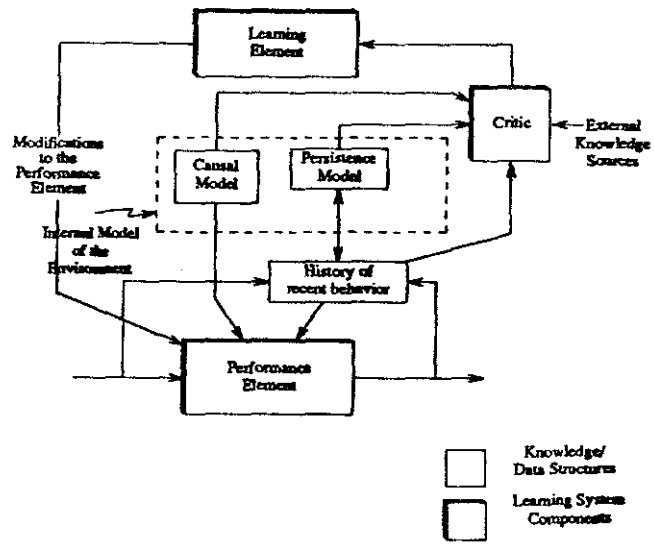
(d) Computational resource and time constraints.

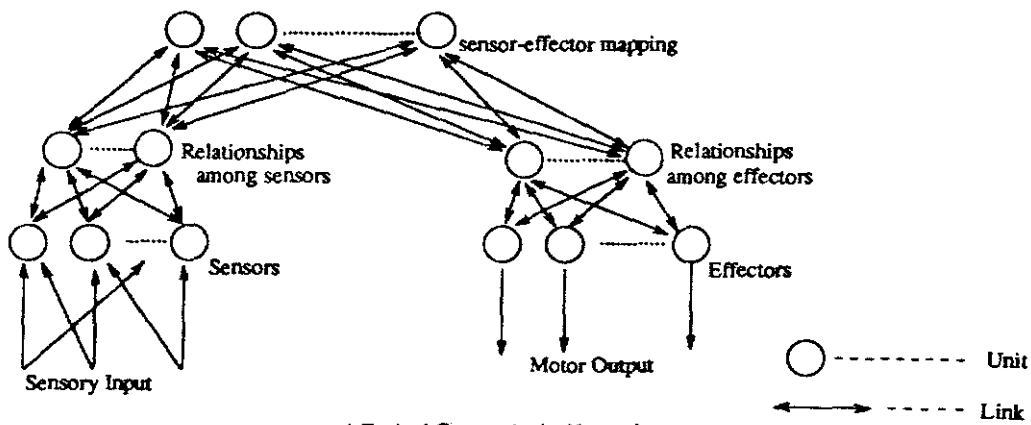
Solving DDPs requires making quick and intelligent decisions. Time can be preserved during decision-making if search is controlled. Search control heuristics are acquired via learning. These heuristics must be explicitly represented as a body of knowledge that can allow simultaneous, intelligent comparision of alternative actions. Learning, like decision-making, must be quick. Such a requirement points towards a parallel, adaptive problem-solving architecture.

SMALL Architecture

The figure below shows a knowledge-level description of a learning and problem-solving architecture that embodies all the bodies of knowledge entailed by the preceding analysis of the strategy-learning task. The performance element contains the heuristics for generating and pruning decision choices; the critic implements the credit-assignment policy of the system; and, the learning element carries out the actual modification of the performance element. Learning episodes are delimited by the arrival of feedback. The history of recent actions maintains a memory of events and actions during such an episode. Together, the causal model and the persistence model represent an internalized model of the problem solver/learner's environment.

The constraint of parallelism and adaptiveness are good reasons to consider connectionist techniques. We first discuss the general classes of networks and techniques, and then describe a realization of the above architecture in terms of such networks.





A Typical Connectionist Network

All the layers besides the sensors are optional. Forward links may bypass intermediate layers and there may be mutual excitatory (positive) or inhibitory (negative) connections within each layer. Links may only be unidirectional in some cases. Some models have complete connectivity between adjacent layers whereas others do not. Units other than sensors and effectors are also known as hidden units or abstract feature detectors. Single-layered ANSs only have one set of weights, which may serve as interconnections among sensors in case there are no effector units, or as links from sensors to effectors otherwise.

Connectionist Methods.

"Connectionism" is the name given to a large class of techniques that rely on network-like data structures composed of computationally active elements for representation of knowledge. Complex associations and complex behaviors emerge from asynchronous, concurrent activity of simple elements. The elements are richly interconnected, and their activities are influenced by the activities of their neighbors. This model of problem solving and learning is variously known as the **parallel distributed processing (PDP)** model, **neural networks** or **artificial neural systems (ANSs)**. A comprehensive review of the paradigm can be found in the references [1, 7, 8].

What are ANSs?

Artificial Neural Systems (see figure) are networks of simple elements. Some designated units act as sensors, so that input to the system can be supplied by influencing the states of these units. Yet others act as effectors, so that the state of these units can be used to direct external actions. The remaining units (appropriately called *knowledge atoms* by Smolensky [10]) capture the relationships among sensors, between sensors and effectors, and among effectors. The global short-term memory of the system is captured locally by the state of activation of each unit. The long-term memory of the system lies in the strength of interconnection (typically represented numerically by a weight) between units. Learning in these systems occurs by the modification of these weights.

Connectionist Primitives.

Specific network configurations for representing specific types of primitive relationships between variables have been developed recently. Those discussed and used in this paper are: **(I) pattern classifiers**, which are capable of judging whether or not a pattern of inputs is a member of a class; **(II) pattern associators**, which learn (possibly bidirectional) associations between input and output patterns; **(III) autoassociators**, which are capable of learning how to complete partial patterns of input, based on the memory of patterns seen in the past; and, **(IV) competitive networks**, which use units with mutual inhibitory connections, in order to select the most strongly activated unit.

Learning in ANSs.

A **connectionist learning paradigm** is a method of modifying weights in response to a pattern of changing activations. Rumelhart et al. [9] have developed the *back-propagation* algorithm for learning in pattern classifiers. Kosko [6] has developed the Adaptive Bidirectional Associative Memory (ABAM) framework for learning pattern associations; Kohonen [5] has developed the Learning Vector Quantization (LVQ) framework for self-organizing autoassociative memories; and, Grossberg [4] has developed the Adaptive Resonance Theory (ART2) framework for competitive learning.

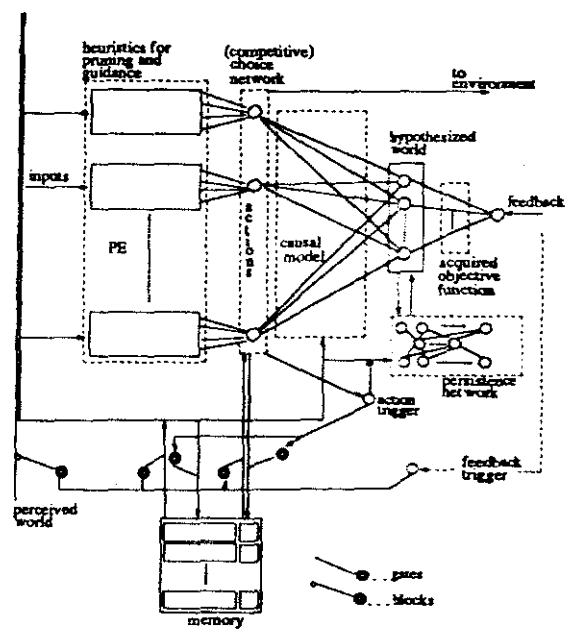
Functionally, pattern classifiers can be applied for inductive concept learning, decision-making, and function approximation. Pattern associators can be used for learning multivariate mappings, storing relational data, and for storing spatio-temporal patterns. Autoassociative networks can be used for optimization, pattern completion, clustering, and as content-addressable memories. Competitive learning mechanisms can be used for choosing among mutually exclusive activation patterns.

ANSs for Strategy Learning.

Although there are not many connectionist systems for strategy learning, this approach has some distinct advantages over other approaches. In particular, decision making under partial information is easier. Connectionist networks exhibit spontaneous generalization (transfer of learning to situations involving similar values for the input parameters). They are quite suitable for storing associations between situations and actions, and have the desirable properties of asynchronous operation and parallel consideration of several decisions. The disadvantage, on the other hand, is that all knowledge is implicit and cannot be translated into other representations. Therefore, there is a *trade-off of flexibility for performance* between connectionist and conventional representations.

ANS-based Implementation of SMALL

A schematic description of a connectionist realization of the SMALL model is shown in the figure. The part of the system marked PE (performance element), together with the choice network, is responsible for choosing a decision in response to an input. The PE is implemented using a pattern classifier network, and the choice network by a competitive activation network. The causal model is realized in a bidirectional associative memory. The perceived value of X becomes available on the lines marked input. The projected or desired values of X become available on units labeled hypothesized world. The internal model of F is realized by another pattern classifier network shown in the figure as acquired objective function. The networks for the persistence model and the memory of recent decisions are realized using autoassociative networks. Numerous triggers and gates are instrumental in controlling the gating and blocking of various interconnections.



Schematic of ANS Implementation

Operation of the ANS ensemble.

The selection and execution of actions continues until feedback is received. The arrival of feedback is signaled by firing the feedback trigger. The feedback trigger disables perception and memorization of decisions. The feedback is propagated through the objective function network to the nodes of hypothesized world model. Thereon, the feedback is apportioned to nodes in the action network by back-propagation through the causal model. Following this, the memorized decisions whose effects still persist are recalled. The event part of the memory is gated to the input lines, and the action part is used to identify the particular portion of the heuristic selection network that will undergo learning. Feedback is multiplied by the persistence of the recalled decision, and then back-propagated through the PE. Ideally, this process of recall should proceed in decreasing order of persistence. However, sorting is hard to implement in connectionist networks. Therefore, the recall proceeds in the opposite order of storage. Once the learner has run through all the stored decisions, it resets the memory. The feedback trigger is reset, and the system proceeds with event-directed (and goal-modulated) decision-making once again.

In the load balancing problem, the process described above realizes the workload distribution policy. Since the system can adapt to changing workload, and since the policy decisions are determined at run-time, the system can be called an adaptive, dynamic load-balancing system. In the next section, we show how the policy is revised by a flexible realization of the critic (the component of the SMALL model responsible for implementing the credit assignment policy).

Knowledge Acquisition

Learning involves *credit-assignment*: knowledge is adapted in response to feedback in the form of reinforcement. As shown above, the credit-assignment process must make use of the acquired objective function, the causal model, and the persistence model. In nonstationary environments, this knowledge is likely to change with time. However, this change is much too slow to warrant continuous learning. On the other hand, it is important enough to warrant the provision of an adaptive critic [3] — it should be possible to revise the bodies of knowledge used by the critic by means of occasional off-line learning. Also, since this knowledge is not provided to the system initially, there is a need for a learning mechanism to acquire it automatically.

The learning of these bodies of knowledge occurs as a *phased learning* process. Learning by experimentation is employed, wherein the learner generates typical learning scenarios, and compares the projected outputs with the observed outputs. The problem of generating hypothetical learning scenarios in load balancing is called *synthetic workload generation*. It is well-known that learning is facilitated by the presentation of examples in which the effect of change in each component of input can be analyzed independently. Thus, "pure" workload generation can simplify learning of workload-dependent quantities. We have recently finished the development of SMALL workload-generation system.

There are three phases of the meta-level learning process — one for each of the bodies of knowledge involved during credit assignment. The objective function is acquired first through workload characterization. The causal model is acquired next by associating the load-balancing decisions and their effects under specific workload conditions. The only information about other processors is the workload information. This causes a major reduction in the amount of information that needs to be communicated. The persistence model is acquired next by studying the decay in persistence of effects following a randomly chosen action, and by repeating this association learning process. Thus, the overall training paradigm reduces the complexity of the *horizontal information compression* by spreading the compression over three phases.

Further reduction in complexity can be attained by viewing the associations between large vectors, as a combination of several subassociations between smaller vectors. Thus, the *vertical information compression* can be spread over learning of these subassociations, followed by learning their composition. The phases of the meta-level learning process are discussed next.

Acquiring the Objective Function.

The objective function value is measured under controlled inputs drawn from a wide variety of task classes. The task remains fixed for each experiment. In case of load balancing, the response time of several representative jobs is measured under varying workload conditions. We have carried out an approximate analysis of the relationship between workload, task parameters, and response time, in order to arrive at a learning strategy that acquires the complex workload measure incrementally.

Acquiring the Causal Model.

Learning by experimentation is employed in this phase. The learner randomly perturbs the environment, and forms an association between the action and the variables affected by the random, controlled action. In case of load balancing, deterministically chosen balancing decisions are performed under controlled workload conditions, and the response time is studied over a variety of tasks. Controlled experimentation continues until the learner can predict the effects of an action (such as, the drop in local load following a migration decision) with reasonable accuracy.

Acquiring the Persistence Model.

Several controlled experiments, each involving a particular member of X, are carried out. Random, controlled actions are performed one at a time. Only those members of X having a causal connection with the chosen action are studied. The environment is allowed to reach quiescence following the performance of an action. The time for quiescence is recorded. The learner attempts to acquire a mapping of the form:

$$\text{Event} \times A \times X \rightarrow \text{Time}.$$

Following these three phases, the system is ready for object-level learning of heuristics by temporal and structural flow of feedback, as already explained. Thus, domain knowledge gets translated into design biases in the construction of meta-level learning networks. However, the volume of prior knowledge required is minimal and the system can bootstrap from very little knowledge of the domain, and the problem solving and learning environment. It can improve its knowledge at both the object level and the meta level. The trained network can make quick and intelligent decisions, in an asynchronous environment. The ill-posedness of the problem is cured by learning an objective function, and the delay in feedback is countered by incorporating persistence and memory modules into the knowledge-level architecture.

Conclusions

Our analysis of the complexity of strategy-learning problems in dynamic, real-time environments, has led to the epistemological characterization of the task, and top-down development of a knowledge-level architecture. Not only have we designed a methodology for tackling problems resulting from delayed feedback, we have also developed a principled training paradigm, which can be adapted for several types of implementations. A connectionist implementation of the proposed model for solving a representative problem has been used to illustrate our approach. We have attempted to advance the research in load-balancing by developing an adaptive approach to workload characterization and distribution, which is sensitive to the workload, architecture, and task requirements of a particular installation. And, we have attempted to advance the research in machine learning by presenting a *general methodology* for acquiring strategies in *complex domains* under *difficult learning situations*.

References

[1] *DARPA Neural Network Study*. Lexington, MA: Lincoln Laboratory, Massachusetts Institute of Technology, July, 1988.

[2] R. Alonso, "The Design of Load Balancing Strategies for Distributed Systems," in *Future Directions in Computer Architecture and Software Workshop*, Seabrook Island, SC, pp. 1-6, May 5-7, 1986.

[3] A. G. Barto, R. S. Sutton, and C. W. Anderson, "Neuronlike Adaptive Elements that can Solve Difficult Learning Control Problems," *IEEE Trans. on Systems, Man and Cybernetics*, vol. SMC-13, pp. 834-846, 1983.

[4] G. A. Carpenter and S. Grossberg, "ART 2: self-organization of stable category recognition codes for analog input patterns," *Applied Optics*, vol. 26, pp. 4919-4930, December, 1987.

[5] T. Kohonen, *Self-Organization and Associative Memory*. Springer-Verlag, (2nd ed.) 1988.

[6] B. Kosko, "Adaptive Bidirectional Associative Memories," *Applied Optics*, vol. 26, pp. 4947-60, December, 1987.

[7] Richard P. Lippmann, "An Introduction to Computing with Neural Nets," *IEEE ASSP magazine*, pp. 4-22, April,1987, (suggested by Peng).

[8] D. D. Rumelhart, G. Hinton, and J. L. McClelland, "A General Framework for Parallel Distributed Processing," in *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, ed., D. E. Rumelhart, J. L McClelland and the PDP Research Group. Cambridge, MA: MIT Press, 1986.

[9] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, *Learning internal Representations By Error Propagation*. Institute for Cognitive Science Report 8506, UCSD, September 1985.

[10] P. Smolensky, "Information Processing in Dynamical Systems: Foundations of Harmony Theory," in *Parallel Distributed Processing: Foundations*. MIT Press, pp. 194-281, 1986.