Toward Learning Systems That Integrate Different Strategies and Representations

TR93-22
Vasant Honavar

September 14, 1993

Iowa State University of Science and Technology
Department of Computer Science
226 Atanasoff
Ames, IA 50011

Toward Learning Systems That Integrate Different Strategies and Representations

Vasant Honavar
Department of Computer Science
226 Atanasoff Hall
Iowa State University
Ames, Iowa 50011-1040

Technical Report 93-22, September 1993

This is a draft of a chapter to be included in: *Symbol Processors and Connectionist Networks for Artificial Intelligence and Cognitive Modelling: Steps toward Principled Integration*, Honavar, V., & Uhr, L. (ed)., New York: Academic Press (1994).

Toward Learning Systems That Integrate Different Strategies and Representations

Vasant Honavar
Department of Computer Science
226 Atanasoff Hall
Iowa State University
Ames, Iowa 50011-1040

This is a draft of a chapter to be included in: *Symbol Processors and Connectionist Networks for Artificial Intelligence and Cognitive Modelling: Steps toward Principled Integration*, Honavar, V., & Uhr, L. (ed)., New York: Academic Press (1994).

Abstract[1]

An understanding of learning – the process by which a learner acquires and refines a broad range of knowledge and skills – is central to the enterprise of building truly adaptive, flexible, robust, and creative intelligent systems. Significant theoretical and empirical contributions to the characterization of learning in computational terms have emerged from research in a number of disparate research paradigms. The limitations of individual paradigms and of particular classes of techniques within each paradigm are beginning to be recognized. Converging lines of evidence from multiple sources, both theoretical as well as empirical, suggest that artificial intelligence systems, in order to be able to deal with complex tasks such as recognizing and describing 3-dimensional objects, or communicating in natural language, must be able to effectively utilize a range of learning algorithms operating with an adequate repertoire of representational structures. This paper draws on a broad range of research on learning in artificial intelligence, connectionist networks, statistical and syntactic methods in pattern recognition, and evolutionary models to identify the similarities and differences among, strengths and limitations of, and promising areas for cross-fertilization between, the different paradigms.

1 Introduction

The ability to learn from experience is an essential attribute of an intelligent entity. No computational account of cognitive processes can be complete without a detailed specification of structures and processes that support *learning*. Over the past few decades, research in several paradigms has attempted to offer accounts of learning in computational terms. This section provides an overview of some definitions and taxonomies of learning, to lay down a context for the discussion of different learning paradigms that follows. Subsequent sections

[1]This work was partially supported by the Iowa State University College of Liberal Arts and Sciences.

briefly summarize each major paradigm that has contributed a significant body of theoretical and empirical insights concerning learning, and identify several interesting areas for cross-fertilization of techniques and ideas emerging from them.

1.1 Learning Defined

Learning may be (informally) defined as: the process of acquisition of new knowledge; the organization of the acquired knowledge into effective representations; the development of perceptual, motor, and cognitive skills; and the discovery of new facts, hypotheses, or theories about the world through exploration, experimentation, induction, deduction, or abduction. For a range of related definitions of learning from the perspective of artificial intelligence and cognitive science, see (Uhr, 1973; Michalski, 1993, Simon, 1983).

Researchers in machine learning often use a definition of learning proposed by Simon (1983) that is motivated by pragmatic considerations. Simon defines learning as any process by which a system quantifiably *improves* its *performance* on a task or a set of tasks with experience over time.

1.2 Learning Categorized

A broad range of learning scenarios and performance measures (e.g., speed, accuracy, robustness, efficiency) are conceivable. The learning scenarios can be classified into a number of (not necessarily disjoint) categories (Uhr, 1973; Michalski, 1993) illuminating the richness of the phenomenon. Most such taxonomies include *rote learning*, *learning from instruction*, *learning from examples (inductive learning)*, *deductive learning* (typically using explanations), *learning from analogy*, *learning by experimentation and discovery*. We can also classify learning based on the general class of computational or mathematical techniques used into *connectionist, symbolic, statistical, syntactic*, and *evolutionary* approaches. Other taxonomies based on other classification criteria are possible. Such taxonomies will without doubt get refined further as our understanding of the computational aspects of learning evolves as new research directions are pursued and new theoretical and experimental results become available.

Any given system may use several types of learning in concert e.g., a system that acquires basic facts by rote memorization can employ inductive techniques to generalize the acquired knowledge, or to identify new situations in which a particular piece of stored knowledge is applicable through deductive reasoning. Complex tasks such as recognizing and describing 3-dimensional objects, and communicating in natural language, almost certainly require a broad range of learning mechanisms operating with a rich panoply of representations.

1.3 Learning and Representation

Central to most forms of learning are the processes of incremental acquisition, modification, consolidation, and adaptation of adequate internal representations of the environment in which the learner operates. Learning (or evolution) must provide the representational structures that perception, cognition, and action utilize and modify (through learning). Indeed, it can be argued that many learning tasks entail the discovery of useful mappings between representations. For example, learning to describe 3—dimensional objects requires the identification of a suitable mapping from the 2-dimensional visual image of the scene to words or sentences in a language that is expressive enough to describe the 3—dimensional structure of the object at a level of detail demanded by the tasks that the learner has to perform. This involves the search of a space of possible mappings permitted by the particular representations in question.

The amount of search effort necessary is a good measure of the difficulty of learning, and this is invariably a function of the representations at the disposal of the learner. Recent results in computational learning theory (Natarajan, 1992) reinforce this intuition. This suggests that successful learning in complex environments such as those that an autonomous robot or a child has to interact with require an adequate armamentarium of representations to choose from as well as learning algorithms that can successfully exploit the strengths of available representations and/or readily and efficiently transform between representations as necessary.

In addition to the search effort (or the time) necessary to learn a task, there are other considerations that might affect the choice of representations and the learning algorithms. For example, if the learner is to successfully cope with complex, occasionally unpredictable environments, it is essential that the learning algorithms and the representational structures be robust in the presence of uncertainty and noise as well as the gradual degradation and occasional failure of system components.

We will repeatedly return to the role of representation in learning later in the paper when we examine the various learning paradigms in detail.

1.4 Learning Paradigms

Most of the research in computational approaches to learning can be traced to one of the following paradigms:

- *Symbol processing* systems or traditional artificial intelligence (AI) (Shavlik & Dietterich, 1990; Buchanan & Wilkins, 1993)

- *Connectionist networks* or neural networks (Kohonen, 1989; Rumelhart et al., 1986; Carpenter & Grossberg, 1991; Gallant, 1993; Kung, 1993)

- *Statistical pattern recognition* (Duda & Hart, 1973; Fukunaga, 1990; Devijver & Kittler, 1982)

- *Syntactic pattern recognition* (Fu, 1982; Miclet, 1986; Pavlidis, 1977 Gonzalez & Thomason, 1978)

- *Genetic algorithms* and evolutionary programming (Holland, 1975; Goldberg, 1989; Koza, 1992)

Research on learning within each of these paradigms has produced a large collection of techniques, empirical and theoretical results, as well as some understanding of the limitations of particular classes of techniques. In addition to the paradigms enumerated above, recent work in *computational learning theory* (Valiant, 1984; Natarajan, 1992) has begun to shed light on some key theoretical questions in learning. In the sections that follow, we will compare the approaches to learning offered by different paradigms, highlight their relative strengths and weaknesses, and explore some possible directions for the synthesis of the best aspects of different paradigms toward the long-term goals of crafting truly general-purpose, adaptive, flexible and creative intelligent systems.

2 Learning in Symbol Processing Systems

The symbol processing approach to the design of intelligent systems was summarized by Newell (1980) and Newell & Simon (1972) in terms of what they called the *physical symbol systems* and by Fodor (1976) in terms of what he called the *language of thought*. In this framework, perception and cognition are tantamount to acquiring and manipulating symbolic representations. Models of intelligent systems developed within this framework typically are (but do not have to be) based on the von Neumann serial stored program model of computation. Popular interpretations of this definition are often overly restrictive, and appear to exclude (for no good reason) systems that perceive, learn, and reason with iconic or analogic representations, or using numerically-encoded probabilistic or fuzzy inference structures.

2.1 Knowledge Representation and Inference

Most of the work based on the symbol processing approach to AI has concentrated on the development of complex domain-specific, knowledge intensive programs (e.g., the so-called *expert systems*). Elaborate knowledge-bases are constructed through a process of *knowledge engineering* (Waterman, 1985) – a complex task that entails making explicit the knowledge and working methods of the human expert, which are usually implicit and difficult to characterize. That is, the human expert must have learned, and be able to make explicit, all the knowledge needed, rather than having the system learn. Knowledge engineering is a difficult and tedious enterprise because experts are often unable to translate the mental processes that they use in solving problems in their domains of expertise into a sufficiently detailed, perfectly-defined, set of rules or procedures.

Use of detailed, explicit knowledge encoded in the form of abstract symbolic descriptions allows such systems to attack *high-level* cognitive tasks while circumventing the difficulties involved in arriving at the knowledge representations starting with raw sensory data. Problems are formulated in terms of searching suitably represented *state spaces* (Ginsberg, 1993; Winston, 1992). Inference using knowledge encoded in a knowledge base usually involves explicitly rule-driven symbol manipulation using production rules (of the form *if* $\mathbf{x}$, *then* $\mathbf{y}$), logic programming, or programs written in a high-level language (such as LISP). Naturally, such systems are prone to brittleness in domains in which it is difficult to encapsulate the relevant knowledge into a knowledge base in the form of a manageable set of rules and symbolic descriptions with sufficient precision, or when the dynamic nature of the task environment calls for the ability to acquire new knowledge and skills as necessary. The difficulties encountered in knowledge engineering in complex domains has prompted recent interest in machine learning as a means of automating the process of knowledge acquisition.

2.2 Learning as Inference + Memorization

Learning in symbol processing systems primarily involves acquisition of, and inference using, knowledge represented in the form of complex symbolic data structures, and the memorization of the results of inference in a form suitable for use in the future. In short, *Learning = Inference + Memorization* (Michalski, 1993). This takes place in the context of background knowledge that the learner has, the environmental input, and the goals or needs of the learner. Following this view, much of the work on learning in symbol processing systems can be categorized in terms of the primary *inference mechanisms* used or the general types of *knowledge transformations* that the input knowledge is subjected to during learning.

Rote learning involves direct memorization of facts as they are received from the environment and involves little inference on the part of the learner. *Learning by instruction* involves *selection* and occasionally *reformulation* (typically by means of syntactic transformations) of the received knowledge. Significant use of inference is employed in various forms of *analytic* and *synthetic* learning. These are discussed in some detail below.

2.3 Analytic Learning Using Deduction

Analytic learning in its pure form uses *deductive inference* (e.g., truth-preserving inference rules of first-order logic such as *modus ponens* and the *resolution principle*, *mathematical induction*) to transform knowledge into a form that is useful for efficient performance of tasks in a given environment. A commonly used analytical learning strategy is *explanation-based learning* (Minton et al., 1990). For example, given a solution to a particular problem in integral calculus (perhaps discovered with time-consuming search of a suitably represented state-space), explanation-based learning mechanism generates an explanation using background knowledge (general rules of integration) showing how the solution deductively follows from what the system knows. The resulting explanation is then used to reformulate the given solution to the particular problem into a form that makes it possible to recognize and solve similar problems efficiently in the future (for example, by reducing the amount of search effort needed to come up with a solution because the necessary knowledge now is in a more usable or *operational* form). Other related forms of analytic learning include deductive derivation of abstractions or generalizations using facts provided by the environment and the background knowledge.

2.4 Extending Analytic Learning

If the background knowledge is incomplete, inconsistent or imprecise, explanation generation, the key component of explanation-based learning cannot proceed without postulating changes in background knowledge. An interesting possibility that is worth exploring is to treat the knowledge base as though it were non-monotonic and the derived explanations as though they were tentative and use a variety of non-deductive learning strategies for hypothesizing candidate revisions of the background knowledge. Other possibilities include non-deductive generalization of explanations as hypotheses to be validated by further experimentation with the environment. Methods developed in the context of syntactic pattern recognition (see below) such as those based on distance measures for structured templates or grammar inference as well as connectionist learning models (see below) offer additional sources of ideas (e.g., adaptation of learned explanations to new situations as hypotheses to be tested) for extending the power of current deductive learning systems.

2.5 Synthetic Learning Using Induction

Induction is the primary inference mechanism used in synthetic learning. Unlike deduction which can lead to no fundamentally new knowledge (because all inferences logically follow from the assumed background knowledge and the given facts), inductive inference allows creation of new knowledge. A typical example of an inductive learning task involves the learning of an unknown concept C_T given a set of examples and/or a counter-examples. Each *example* is an ordered pair (I_k, M_{C_T}) where I_k is an *instance* represented in the chosen *instance description language* and M_{C_T} indicates the membership of the instance I_k in the concept C_T. Thus, in the case of binary concepts M_{C_T} takes binary values. The system is also equipped with a *concept description language* which is used to express *candidate concept descriptions* $C_i \in C$ (where C is the set of all concept descriptions expressible in the language)) and a means of determining the membership of an instance I_k in a given concept C_i.

2.6 Representational and Inductive Biases in Concept Learning

In the framework outlined above, the problem of concept learning essentially reduces to a search for a concept description $C_l \in C$ that *agrees with* or *approximates* the unknown concept C_T on the set of examples S. When it is the case that multiple candidate concept descriptions qualify this criterion, additional preference criteria (e.g., simplicity) have to be used to choose one such C_l. The choices of the concept description language and the instance description language constitute important representational commitments. The size of the search space that the learner has to deal with is a function of these choices. Computational learning theory (Valiant, 1984; Natarajan, 1992) attempts to quantify the complexity of learning in terms of the number of samples and the running time of the learning algorithm. The feasibility of learning depends critically on the choice of languages (representations) used for describing instances and concepts.

Since concepts are essentially descriptions, definitions, or tests for membership in a class, of sets of instances, a variety of mechanisms can be used to bias the way the search through the space of candidate concepts is conducted. Thus, one may *generalize* from specific instances toward candidate concept descriptions using a number of inductive inference rules. For example, if the concept description language uses first order logic predicates, inductive inference rules might include turning constants into variables, adding disjuncts, dropping conjuncts, extending the domain of variables, etc. (Michalski, 1984). Alternatively, one may *specialize* from overly general descriptions so as to exclude counter-examples of the concept. It is possible to employ bi-directional search of the space of concept descriptions so that both generalization and specialization go hand in hand (Mitchell, 1982). Other representations allow similarly motivated generalization and specialization operations.

2.7 Extending Inductive Learning in Symbol Processing Systems

A variety of techniques developed in the context of statistical pattern recognition and connectionist networks can be adapted to overcome some of the limitations (notably, brittleness) of learning in symbol processing systems. Generative learning methods developed for connectionist networks (Honavar & Uhr, 1988; 1989a; 1989b; 1993) (see below) can be extended to a variety of structured symbolic representations such as strings, trees, and graphs (Honavar, 1992). Shavlik (1993) explores such hybrid techniques in the context of learning classification rules. Gennari, Langley, and Fisher (1990) discuss the use of statistical criteria to guide incremental cluster formation in concept learning systems.

Inductive learning need not be limited to concept learning from examples. Different amounts of background knowledge may be used to supplement example-driven learning and to facilitate active exploration of conceptual spaces. Some background knowledge may be provided implicitly in terms of choice of a range of representations with different representational biases and different degrees of expressive power as well as the ability to switch between, choose among, and intelligently navigate through them are worth exploring (see Honavar, 1992c; Yang & Honavar, 1993 for an example of such a system for learning to recognize and describe 3-dimensional objects). Also of interest are mechanisms that can extend the concept description language as needed – possibly with the aid of conceptual clustering to detect and name regularities, deduce, as well as are evolutionary methods that generate and evaluate new or more useful representational primitives (see below).

2.8 Other Forms of Learning in Symbol Processing Systems

Abduction involves the generation of candidate hypotheses for observed or otherwise given facts in terms of background knowledge and additional assumptions or hypotheses. The utility of such hypotheses is primarily in terms of guiding the search for useful theories of particular domains of interest. Abduction when used with analytic explanation-based learning can help extend background knowledge through the proposal and evaluation of candidate explanations. Unlike induction and deduction, the use of abduction in machine learning has not been widely explored, yet abduction appears to play a central role in learning and discovery.

Analogy refers to a mapping between two entities (objects, events, problems, behaviors, etc.). Relatively little is known about the formation of analogical mappings or analogical inference (but see Gentner, 1993). Analogical inference appears to be an integral part of human reasoning so we mention it here for the sake of completeness.

Discovery involves the search for qualitative and quantitative laws, testable hypotheses, explanations, and descriptive as well as prescriptive theories for particular domains or sets of domains (Honavar, 1993; Uhr, 1973). In its most general form, it can use analytic as well as synthetic learning of all types. Only the simplest forms of learning by discovery have been examined to date (Langley & Zytkow, 1990; Lenat, 1982; Shrager & Langley, 1990), leaving a broad range of possibilities to be explored.

3 Learning in Connectionist Network Systems

Connectionist networks or *artificial neural networks* are massively parallel, shallowly serial, highly interconnected networks of relatively simple computing elements or *neurons* (Grossberg, 1991; Hinton, 1990; Kohonen, 1989; Gallant, 1992; Kung, 1993). Much of the attraction of connectionist networks is due to their massive parallelism that is amenable to cooperative and competitive computation, potential for limited noise and fault-tolerance, discovery of simple, mathematically elegant learning algorithms for certain classes of such networks, and to those interested in cognitive and brain-modelling, their (albeit superficial) similarity with networks of neurons in the brain.

3.1 Representation and Computation in Connectionist Networks

The input to an n-input neuron can be represented by a pattern vector $\mathbf{X} \in \Re^n$ or in the case of binary patterns, by a binary vector $\mathbf{X} \in [0,1]^n$. Each neuron computes a relatively simple function of its inputs and transmits outputs to other neurons to which it is connected via its output links. A variety of neuron functions are used in practice. The most commonly used are the *linear*, the *threshold*, and the *sigmoid*. Each neuron has associated with it a set of parameters which are modifiable through learning. The most commonly used parameters are the so-called *weights*. The weights associated with an n-input neuron i are represented by an n-dimensional weight vector $\mathbf{W}_i \in \Re^n$. In the case of a linear neuron, the output o_i in response to an input pattern $\mathbf{X}$ on its input links is given by the vector dot product $\mathbf{W}_i \cdot \mathbf{X}$. In the case of a threshold neuron, $o_i = 1$ if $\mathbf{W}_i \cdot \mathbf{X} > 0$ and $o_i = 0$ otherwise. For a sigmoid neuron, $o_i = 1/(1 + e^{-\mathbf{W}_i \cdot \mathbf{X}})$. In each of these cases, usually, one of the n inputs is held constant and the corresponding weight is called the threshold.

The functions listed above by no means exhaust the possible choices for neuron functions and a wide range of other possibilities exist. The representational and computational power of such networks depends on the functions computed by the individual neurons as well as the architecture of the network (e.g., the number of neurons and how the neurons are connected). In general, a layered *feed-forward* network with n inputs and m outputs can represent some subset of the possible mappings from $\Re^n$ to $\Re^m$. Such networks find application in data compression, feature extraction, pattern classification and function approximation. A great deal is known about the representational power of different classes of feed-forward networks. For example, a 2-layer feed-forward network (not counting the input neurons that simply transmit the components of the input vector $\mathbf{X}$) with a *sufficiently large* (finite) number of sigmoid neurons in the first layer and linear neurons in the output layer can approximate to arbitrary accuracy, arbitrary continuous functions on bounded closed subsets of $\Re^n$. When feedback loops are included (thereby allowing the network's output at a given time step to be influenced by its output at a previous time step), it can produce complex temporal dynamics. Such networks find use in applications such as robot motion control and approximation of regular grammars from examples. Connectionist networks become Turing-equivalent in their computational power if we allow the networks to grow arbitrarily large.

3.2 Learning as Parameter Modification

Much of the research on learning in connectionist networks has tended to focus on algorithms for changing the modifiable parameters (weights) in networks with a certain a-priori chosen network architecture using samples of the function to be approximated or patterns to be classified. In other words, the task of the learning algorithms is to find a set of weights that yields a satisfactory approximation of the unknown function on the given samples (The expectation is that the network will generalize well on samples not seen during training – more on this later). This is fundamentally a search or optimization problem and so a variety of linear and non-linear optimization methods (gradient-descent, simulated annealing, etc.) can be used in this context. For details of such algorithms, the reader is referred to (Hinton, 1990; Kung, 1993).

Many connectionist learning algorithms use some form of error-guided search (e.g., changing each modifiable parameter in the direction of the negative gradient of a suitably defined error measure with respect to the parameter of interest. A commonly used error measure in function approximation applications is the *mean squared error* between the desired and actual network outputs. For pattern classification tasks, a number of different error measures motivated by statistics and information theory (e.g., loss functions, cross-entropy) are possible candidates. For data compression or feature extraction, other error measures can be used to map high-dimensional input patterns to feature vectors in a lower dimensional space with minimal loss of information. Common examples of such methods include *competitive learning networks* and *principal component analysis*. It is also possible to introduce additional constraints on the weights in the form of suitably defined additive terms to the primary error measure. For example, generalization performance of networks can be improved by forcing small weights to decay towards zero (Hanson & Pratt, 1990).

Learning by weight modification requires (among other things) an adequate choice of initial network architecture (e.g., a sufficient number of suitably connected neurons). However, learning need not be limited to weight modification alone within an otherwise a-priori specified network.

3.3 Learning by Changing Network Architecture

Learning by weight modification involves searching a high-dimensional space of weights for an acceptable solution that satisfies the desired performance criterion. In order for this to work, an acceptable solution must in fact lie in the space being searched, and the particular search algorithm used must in fact be able to find it. In general, there is no way to guarantee that an ad hoc choice of network architecture would be adequate for learning a mapping implicitly specified in terms of a set of sample input-output pairs. Generative or constructive algorithms (Honavar & Uhr, 1988; 1989a; 1989b; 1993; Honavar, 1992b; Fahlman & Lebiere, 1990) attempt to discover an adequate network architecture for a given task. This involves conducting a search in the space of network topologies (which, unless suitably constrained, can easily get out of hand). Most successful generative algorithms bias the search towards discovering parsimonious networks by letting weight modification do its work and allowing new neurons to be recruited into the network only when the learning curve has flattened out. In addition, they include mechanisms to ensure that a neuron thus recruited in fact reduces the residual error to yield a better approximation. For a discussion of a number of alternative generative learning algorithms for connectionist networks, see (Honavar & Uhr, 1993); and for an example of their use in task-driven construction of multi-resolution representations of 2-dimensional images, see (Honavar, 1992b). The use of network pruning techniques that force redundant or useless weights to gradually decay towards zero and discard useless nodes offers additional interesting possibilities that remain to be explored in conjunction with network growing algorithms.

3.4 Extending Connectionist Learning

Connectionist research on learning (Grossberg, 1991; Kohonen, 1984; Rumelhart & McClelland, 1986; Hinton, 1990; Gallant, 1992; Kung, 1993) has focused, for the most part, on similarity-driven feature discovery, associative storage and recall of binary or analog pattern vectors (i.e., rote learning), learning of input-output mappings between boolean or real-valued vector spaces, and temporal dynamics of systems using input-output behavior sampled in time for prediction and control applications. When connectionist learning techniques are examined alongside the developments in other learning paradigms discussed in this paper, a number of promising combinations present themselves.

3.5 Use of Prior Knowledge With Connectionist Learning

A promising research area is the study of techniques for incorporating prior (possibly domain specific or heuristic) knowledge into connectionist networks. Such knowledge can take a variety of forms. A number of researchers have investigated techniques of initializing a network with knowledge available in the form of propositional rules (Shavlik, 1993). The network's initial knowledge is gradually refined with the use of training examples that fine-tune the weights in the network. Others have investigated the use of topological constraints or inductive biases to guide the search for weights or network architectures into promising regions of the solution space (Honavar & Uhr, 1988; 1989a; 1989b; 1993; Honavar, 1992b). Still others have examined techniques for partitioning the task into more or less independent sub-tasks (using domains-specific knowledge) (Tesauro, 1989; Jacobs, Jordan, & Barto, 1990; Ballard, 1987; Rueckl, Cave, and Kosslyn, 1989; Waibel, 1989) to facilitate learning.

3.6 Connectionist Learning With Structured Symbolic Representations

The applicability of connectionist learning methods to complex tasks such as 3-dimensional object recognition and description and language learning is limited by the representation of input patterns as unstructured numeric vectors. Thus iconic or structural information (e.g., relative spatial arrangement of features in a 2-dimensional image) implicitly encoded in an input pattern is lost when such patterns are mapped in an ad-hoc fashion into numeric vectors to facilitate the use of standard connectionist learning algorithms.

With the exception of some recent work in connectionist architectures for variable binding and unification in slot and filler type representations (Smolensky, 1990; Hendler, 1989; Sun, 1993), very little systematic study has been done on the representation and use (especially in learning) of structured objects such as frames (Minsky, 1975), schemas (Arbib, 1993) and conceptual graphs (Sowa, 1984) in connectionist networks. One possibility is to use structured object representations of the sort developed in the context of syntactic pattern recognition (see below) along with suitably defined similarity metrics with generative learning algorithms. Such systems would extract, abstract, and encode instances of complex structured objects as necessary and use parameter modification algorithms to fine-tune the representations thus constructed (Honavar, 1992a).

The use of finite automata (regular language recognizers) and other more powerful microcircuits in place of the extremely simple processing elements used in connectionist networks suggest other possibilities. A variety of learning algorithms that recruit diverse types of processing modules as needed into connectionist networks using mechanisms similar to those used in generative learning for recruiting from a pool of candidate neurons are worth investigating as well.

4 Learning in Statistical Pattern Recognition Systems

Learning in statistical pattern recognition systems (Duda & Hart, 1973; Devijver & Kittler, 1982; Fukunaga, 1990) is a form of inductive learning. It is typically assumed that the objects in the domain of interest can be distinguished among each other based on a set of measurements. As in the case of connectionist networks, such measurements are usually expressed in the form of a vector $\mathbf{X}$ in an n-dimensional space $\Re^n$.

4.1 Learning by Estimating Density Functions

Learning involves the analysis of the statistical distribution of the measurement vector $\mathbf{X}$ for different classes of objects in the domain of interest using a training set of measurement vectors with known class memberships.

Once the properties of different object classes are learned in terms of the distribution of the measurement vectors in $\Re^n$, classification of a novel sample is conceptually rather straightforward: Simply assign the sample to the class that is most likely given the measurement vector for the sample. This is usually accomplished by defining suitable *discriminant functions* that divide the n-dimensional space into regions that correspond to different classes of objects (or equivalently, patterns). Suppose $P(C_i)$ is the (known, estimated, or assumed) a-priori probability of occurrence of patterns from class C_i; and $p(\mathbf{X}|C_i)$, the class-conditioned density function of the random variable $\mathbf{X}$ given that the corresponding sample belongs to class C_i (this is estimated from samples whose class memberships are known). From Bayes theorem we have, the likelihood that a sample belongs to class C_i conditioned on the measurement vector $\mathbf{X}$, that is, $P(C_i|\mathbf{X}) = p(\mathbf{X}|C_i)P(C_i)/p(\mathbf{X})$ where $p(\mathbf{X}) = \sum_i p(\mathbf{X}|C_i)P(C_i)$. Therefore, we can easily compute $P(C_i|\mathbf{X})$ once we have the estimates for $p(\mathbf{X}|C_i)$ and $P(C_i)$. This is then used to assign the sample corresponding to the measurement vector $\mathbf{X}$ to the class C_k if $\forall j \neq k$ we have $P(C_k|\mathbf{X}) \geq P(C_j|\mathbf{X})$. The computation necessary for classifying an sample can often be expressed in the form of a suitable *discriminant* function $g(\mathbf{X})$ that returns the classification of a pattern corresponding to the measurement vector $\mathbf{X}$.

There are two broad types of statistical learning methods: *parametric* and *non-parametric*.

4.2 Parametric Methods

Parametric methods assume either the mathematical *form* of the class-conditioned density functions $p(\mathbf{X}|C_i)$ for each pattern class C_i, or the discriminant functions $g_i(\mathbf{X})$. *Multivariate gaussian* is a common choice for the form of the density functions. This reduces learning to the estimation of the *parameters* (e.g., mean vector and the covariance matrix in the case of the multivariate gaussian) that characterize the assumed distribution using the training set. *Linear*, *quadratic*, or *piece-wise linear* forms are common choices for the discriminant functions.

4.3 Non-parametric Methods

Non-parametric methods do not assume the form of the underlying distribution of measurement vectors but instead, *estimate* the distribution using the training set. A number of such estimators (based on different notions of what constitutes a good estimate) are available (Duda & Hart, 1973; Devijver & Kittler, 1982; Fukunaga, 1990). A popular example of a non-parametric method is that of nearest neighbors in which a sample is assigned to the class of its nearest neighbor(s) (in terms of a suitable measure of distance) in the training set.

4.4 Extending Statistical Methods

It must be obvious that the efficacy of statistical learning of the sort outlined above is critically dependent on the choice of measurements used to represent the samples in the form of a vector of measurements. Also, the expressive power of this representation is limited if the goal is to describe useful *structural* relationships among the various parts of an object. Clearly, alternative frameworks for knowledge representation are worth investigating for use with statistical learning methods (see below). It must also be pointed out that statistical methods by their very nature rely on the availability of a sufficiently large number of samples to come up with good estimates of the distributions of interest. This limits their applicability in a number of practical learning scenarios where samples are scarce or expensive.

5 Learning in Syntactic Pattern Recognition Systems

Syntactic or *structural* pattern recognition methods (Fu, 1982; Miclet, 1986; Gonzalez & Thomason, 1978; Pavlidis, 1977) emphasize the structural relationships among the components of patterns in the domain of interest. Thus, if the domain of interest consists of curves in a 2-dimensional plane, we can define a finite alphabet V_T, each of whose letters corresponds to a direction. Thus, we can use an 8-letter alphabet to represent 8 discrete directions. Any curve can be represented in discrete form as an ordered sequence of suitable letters chosen from V_T — that is, a string X ($X \in V_T^\star$ — the set of all possible strings over the alphabet V_T). Pattern classes then correspond to subsets of ${V_T}^\star$ that share the desired properties. The structure of complex patterns (wholes) are described in terms of (possibly hierarchical) compositions of (simpler) sub-patterns (parts).

The simplest sub-patterns used to describe patterns or objects in a domain of interest are called *pattern primitives*. The *language* that provides the structural description of patterns in terms of a set of pattern primitives and their rules of composition may be thought of as the *representation language*. The rules governing the composition of primitives into patterns constitute the *grammar* of the representation language. For example, a class of patterns C_i (say houses) may be described in terms of a finite set of easily recognizable pattern primitives and a grammar G_i that specifies the possibly infinite number of patterns of that class.

More precisely, a string grammar is specified as $G = (V_T, V_N, P, S)$ where V_T is the set of *terminal symbols* (primitives), V_N is the set of *non-terminal* or *auxiliary* symbols, P (in general) is the set of *production rules* of the form $\alpha \to \beta$ where $\alpha \in (V_N \cup V_T)^\star V_N (V_N \cup V_T)^\star$ and $\beta \in (V_N \cup V_T)^\star$, and $S \in V_T$ is the *axiom*. The set of all strings derivable using the grammar G is referred to as the language $L(G)$. By placing different restrictions on the form of rules in P, we get different classes (e.g., *regular*, *context-free*, and *context-sensitive*) of grammars.

5.1 Pattern Recognition as Parsing

Recognizing a pattern as a member of the class C_i can be accomplished by *parsing* the *sentence* corresponding to the pattern to determine whether or not it is syntactically (grammatically) derivable using the specified grammar G_i. If the grammar is *regular*, finite state automata can be used to recognize grammatical sentences. Of course, it is well-known that the descriptive power of the finite state languages is weaker than that of context-free and context-sensitive languages. At the same time, the analysis of context-sensitive languages is far more complicated than regular or context-free languages. Various tradeoffs can be made between the descriptive power of the language and the efficiency of syntax analysis. It is often possible to approximate languages of interest by regular (finite state) languages or to use domain-specific syntactic or semantic constraints to facilitate efficient parsing (Fu, 1982).

5.2 Pattern description languages

Representation of patterns by linear strings permits the encoding of only a single relation between the pattern primitives (i.e., a pattern primitive can only precede or succeed another). A number of special languages have been proposed for description of special classes of patterns such as hand-written characters, chromosome images, line drawings, three-dimensional objects, etc. (Rosenfeld, 1981; Pavlidis, 1977; Winston, 1975). When we allow more general classes of n-ary relations, we can define a wide range of high-dimensional pattern grammars. For example, a simple 2-dimensional generalization of string grammars gives *array grammars*. Other generalizations include *web grammars*, *tree grammars*, *plex grammars* and *graph grammars*.

5.3 Learning as Grammar Induction

Assuming that a satisfactory solution for the selection of pattern primitives and the task of recognizing grammatical sentences can be found for a given application, learning to recognize classes of patterns reduces to learning the corresponding grammars (or equivalently the corresponding automata).

Suppose G is an unknown grammar and $L(G)$ the corresponding language—that is, the set of all sentences derivable from G. The problem of inferring G given a finite subset of $L(G)$ (and possibly the complement of $L(G)$) is called the *grammar induction* or *grammar inference* problem. Various criteria can be used to evaluate the inferred grammar, most of which take into account the degree to which the inferred grammar agrees with the unknown grammar on strings that were not provided during learning. Other criteria for such evaluation might include various measures of complexity of the inferred grammar. Space does not permit a detailed survey of grammar inference here (see Fu, 1982; Gonzalez & Thomason, 1978; Miclet, 1986; Natarajan, 1992; Parekh & Honavar, 1993b for details). For a discussion of recent work on the efficient inference of a regular grammar G given a set of samples from $L(G)$ and the teacher's responses to a finite number of membership queries on strings generated by the learner, see (Parekh & Honavar, 1993a; 1993b); and for examples of connectionist approaches to grammar induction, see (Omlin & Giles, 1993).

5.4 Attributed Grammars

In many practical problems, it becomes necessary to supplement classical grammars which are purely syntactic in nature with *semantic* information. Borrowing from developments in the theory of programming languages (Knuth, 1968) and linguistics, we can add semantic attributes to each substructure–in the case of grammars, to each terminal or auxiliary element as follows: Suppose $G = (V_T, V_N, P, S)$ is a context-free grammar with the production rules P of the form $n \rightarrow \delta$ where $n \in V_N$, and $\delta \in (V_N \cup V_T)^+$. With each element $x \in (V_N \cup V_T)$, we associate a finite set of attributes $A(x)$. Each attribute $a \in A(x)$ takes values from a (finite or infinite) set of values – i.e., its domain D_a from which one value is selected for each occurrence of x in a derivation tree for some sentence derivable using G.

More precisely, if P consists of m production rules, and if the kth production rule is of the form $x_{k_0} \to x_{k_1} x_{k_2} ... x_{k_{n_k}}$, where $x_{k_0} \in V_N$, $n_k \geq 0$ and $x_{k_j} \in (V_N \cup V_T)$, $1 \leq j \leq n_k$, then functions f_{kja} defined for all $1 \leq k \leq m$, $0 \leq j \leq n_k$ and $a \in A(x_{k_j})$ map values of certain attributes of $x_{k0}, x_{k1}, ... x_{kn_k}$ into a suitable value of some attribute of x_{kj}. That is, each such function is a mapping from $D_{a_1} \times D_{a_2} ... \times D_{a_t}$ into D_a for some $t = t(k, j, a) \geq 0$ where each $a_i = a_i(k, j, a)$, $1 \leq i \leq t$ is an attribute of some x_{kl_i} for $0 \leq l_i \leq n_k$. These functions therefore can be viewed as attributing *meaning* to strings derived from the application of rules of grammar G.

5.5 Learning Semantics Using Attributed Grammars

Methods of grammar inference can be used to learn the syntactic structure of patterns. However, semantics of sentences play a major role in perception, cognition, and language use. Potentially useful strategies for learning semantics using attributed grammars become apparent once we observe that the task is simply one of learning the functions or mappings $f_{kja} : D_{a_1} \times D_{a_2} ... \times D_{a_t} \to D_a$. The learning of such mappings is precisely what most connectionist network and some statistical learning algorithms do. Thus connectionist learning can be used to provide the needed semantic content for otherwise purely syntactic systems (this is facilitated by the fact that these functions can map (external or internal) analog sensory input to the system (e.g., the visual input via a retina or a camera; internal sensors for energy level) into attribute values associated with symbols in a particular context or from sets of attribute values associated with internal symbols into analog signals to the effectors (e.g., a robot arm) to transmit signals or cause physical changes in the environment. The syntax of sentences in a natural language may be learned and coded in the form of a grammar G and the semantic associations between elements of syntactic categories learned and stored as a set of functional mappings F (whose elements are the functions f_{kja}).

5.6 Pattern Recognition as Template Matching

As a possible alternative to using the rules of a grammar, the process of pattern recognition may be formulated as a process of *template matching* wherein a pattern is matched (using a suitable *similarity metric* or *distance measure* against one or more stored pattern(s) or template(s) or prototype(s) or reference pattern(s) for each of the pattern classes of interest and assigned to the pattern class with the best match. In this case, learning reduces to the acquisition of a suitable collection of templates (and perhaps selection of suitable similarity metrics) adequate for the task.

5.7 Distance Measures for Structured Patterns

If patterns are represented by strings, an intuitively appealing distance measure is the *Levenshtein distance*, originally proposed in the context of error–correcting codes (Levenshtein, 1966). The Levenshtein distance (also called edit distance) from a string α to another string β is measured by the *minimum cost* of transforming α into β. Such transformation is accomplished by a sequence of basic operations such as *substitution* of a letter with another letter from the alphabet, *insertion* of a letter and *deletion* of a letter. Each of the basic operations is assigned a non-negative *cost* and the total cost of the transformation is computed as a weighted sum of the individual costs.

The definition of weighted edit distance defined above for strings is easily extended to trees whose nodes are labeled with the letters of an alphabet: Trees can be viewed as a recursive string of subtrees. Tree—based representations permit efficient encoding of hierarchical relations among successively complex sub-patterns. Quad-trees, oct-trees, and more generally k-trees and pyramids are widely used in computer vision (Rosenfeld, 1984; Uhr, 1987; Tanimoto & Klinger, 1980). This opens up the possibility of using tree grammars and edit distances between trees for learning in such domains.

Distance between two graphs can be defined in a manner analogous to that in the case of strings and trees as the minimum cost of transforming one graph into another. However, unlike the case of strings and trees, the exact computation of such distance between arbitrary graphs is computationally intractable. Practically useful yet computationally tractable heuristic algorithms for computing distance (or, conversely, similarity) between graphs are only beginning to be explored. Another possibility is to investigate restricted subclasses of graphs that lend themselves to efficient distance computation.

It is natural to generalize the idea of distance to arbitrary structured representations (Honavar, 1992a; Goldfarb, 1990). Consider a representational framework $\mathcal{R}$ (e.g., a space of appropriately defined vectors, strings, trees, graphs). Let I_1 and I_2 be objects represented in $\mathcal{R}$. That is, if $\mathcal{R}$ is the space of strings defined over a specified alphabet, say V_T, then I_1, I$_2$ $\in V_T{}^{\star}$. Key to the notion of generalized distance measure is a set of *elementary operations*, each with its associated non-negative cost, that can be used to transform I_1 into I_2. The *generalized distance* between I_1 and I_2, $\Delta_{\mathcal{R}}(I_1, I_2)$ is given by the *minimum cost* (or some heuristic approximation of the minimum cost when an exact calculation of the minimum is not feasible for computational reasons) of transforming I_1 into I_2. This notion of generalized distance allows us to specify classes of inductive learning algorithms for families of representations.

Distance computation algorithms not only provide a degree of mismatch (or conversely similarity) between two objects I_1 and I_2, but also identify the sequence of transformations that would make I_1 identical to I_2. Thus we get for free, the ability to describe how two objects differ from each other. This opens up the possibility of interactive learning systems which can seek and use a richer variety of feedback from a teacher (e.g., pointing out the structures and their inter-relationships missed by the learner) than is typically provided in empirical inductive learning.

5.8 Learning by Acquisition and Modification of Structured Templates

The template matching approach to pattern recognition is reminiscent of exemplar-based or case-based reasoning in artificial intelligence. Thus it is possible to adapt much of the work in exemplar-based learning to structured templates. Learning in this case reduces to the task of acquiring and modifying templates necessary for specific tasks (e.g., pattern recognition). It is rather straightforward to adapt generative or constructive learning algorithms developed in the context of connectionist networks (see above) to add new structured templates through a feedback-guided process of extraction and abstraction of patterns and subpatterns from the environmental input once suitable similarity or distance measures are defined and efficient algorithms for computing such distances are specified (Honavar, 1992a).

Many of the generalization and abstraction algorithms used in symbol processing approaches can find use in the modification of acquired templates. Learning can also modify the non-negative weights associated with particular transformations so as to reduce (increase) the distance between an input pattern and a stored template if the feedback suggests they are similar (dissimilar) as done in the case of some connectionist learning systems.

Another interesting possibility is for learning to extend the set of primitives used by adding potentially useful or meaningful compositions of existing primitive patterns. The primitives so added have an effect analogous to extending the vocabulary of natural language by defining new terms or concepts. Similar *chunking* mechanisms have been explored in symbol processing systems (Laird, Rosenbloom, & Newell, 1986; Uhr & Vossler, 1963), generative learning algorithms for connectionist systems (Honavar & Uhr, 1988; 1989a; 1989b; 1993), and in evolutionary learning systems.

With attributed structured objects (structured objects generated by a (possibly unknown) attributed grammar), it is possible to learn functions that map from sets of attribute values to values of specific attributes. Learning of such functions can be accomplished with many of the similarity-based learning algorithms of symbol processing AI systems or typical connectionist network and statistical learning algorithms.

5.9 Pattern Recognition Using Grammars as Templates

It is possible to combine the template–based and grammar–based approaches to syntactic pattern recognition. For example, a pre-specified or learned grammar G_i for a class of patterns C_i can be treated in a manner analogous to a template for the class. One can then define the distance between a given string γ and the grammar G_i as the minimum cost of transforming the string γ so as to render it syntactically acceptable by the grammar G_i. This distance can be used to assign patterns to one of several candidate classes.

5.10 Stochastic Grammars or Weighted Syntactic Models

Our discussion of grammars has relied on an implicit assumption, namely that every pattern belonging to the language $L(G)$ is equally significant in the definition of the grammar G. In practice however, some patterns are more likely to appear than others. In other words, we need a way to compute the probability of production of a particular pattern X by a given grammar G_i. Then we can, given the a-priori probability of occurrence of the pattern X, namely $P(X)$, compute the likelihood that the grammar G_i produced the observed pattern X, namely $P(G_i)|X$. Given a set of grammars $G_1, ...G_N$ to choose from as possible sources of an observed pattern, we can simply pick G_k such that $\forall j \neq k$ $P(G_k|X) \geq P(G_j|X)$. The theory of stochastic grammars is concerned with defining these probabilities in a consistent, meaningful way.

Assuming that, in general, the probability of application of a production rule at a given stage is independent of the sequence of rules that have been applied up to that stage, we can define a stochastic grammar as $G_S = (V_N, V_T, P, S)$ where each rule $r \in P$ has associated with it, a probability p_r. Inference of stochastic grammars can be accomplished by a two-stage process: First, a non-stochastic grammar G is inferred based on the sample strings from the unknown grammar. Then the probabilities p_r associated with each rule $r \in P$ are estimated from the samples using standard statistical estimation techniques (Fu, 1982). This is a good example of a learning scheme wherein both syntactic and statistical paradigms go hand in hand. Stochastic extensions of attributed grammars offer additional interesting possibilities that remain to be explored.

6 Learning in Evolutionary Systems

Genetic algorithms and classifier systems (Holland, 1975; Booker, Goldberg, & Holland, 1990; Goldberg, 1989), evolutionary programming (Fogel, Owens, & Walsh, 1966), and genetic programming (Koza, 1992) are all related techniques that find use in machine learning. They are all inspired by processes that appear to be used in biological evolution and the working of the immune system. Central to such evolutionary systems is the idea of a population of potential solutions (individuals) that correspond to points in a high-dimensional search space. In genetic algorithms, typically the points in an n-dimensional search space are represented by binary vectors $\mathbf{X} \in [0,1]^n$. For example, such an encoding might represent the state transition table of a finite state machine.

It is assumed that the quality of each candidate solution (or fitness of the individual in the population) can be evaluated in some fashion. Evolutionary systems use some form of fitness-dependent probabilistic selection of individuals from the current population to produce individuals for the next generation. The selected individuals are subjected to the action of genetic operators such as *mutation* and *crossover* that result in new individuals that replace some of the low-fitnes individuals in the population. Mutation operates on a single string and generally changes a bit at random. Thus, a string 11010 may, as a consequence of random mutation, get changed to 11110. Crossover on the other hand, operates on two *parent strings* to produce two *offspring*. With a randomly chosen crossing site of 4, the two strings 01101 and 11000 yield the offspring 01100 and 11001 as a result of crossover. In addition to the simple genetic operators and selection mechanisms outlined here, a large number of alternatives have been studied in the literature (Goldberg, 1989; Michalewicz, 1992).

It can be shown that evolutionary process of the sort outlined above simulates a highly opportunistic and exploitative randomized search algorithm that explores high-dimensional search spaces rather efficiently (Holland, 1975).

Since most learning tasks can be formulated as essentially search problems, it is natural to apply evolutionary algorithms in machine learning. The most straightforward scheme would be to encode the solution space of the learning problem in the form of bit-strings that constitute potential solutions (individuals in the population). This is the approach taken in (Smith, 1980). Holland (1975) takes a different approach: the solution to a problem is a coupled set of rules but the population consists of individual rules (see below).

6.1 Representation and Learning in Classifier Systems

Classifier systems (Holland, 1975) are adaptive *rule-based production systems* in which the rules of the form *if* **x** *then* **y** are restricted to have extremely simple syntax. For obvious reasons, such rules are called *classifiers*.

The condition part of the rule is a fixed length string over an alphabet that is limited to 0, 1, and a don't care. The action part of the rule is a fixed length binary string. The use of fixed length strings permits the use of simple genetic operators to explore efficiently the space of possible rules – provided of course that a good mechanism can be found to evaluate the *fitness* of a rule.

Unlike most traditional expert system models used in artificial intelligence, classifier systems use parallel rule activation – that is, every classifier that has its condition part matched by the messages currently in the working memory is allowed to compete with other such classifiers for the right to fire (i.e., change the working memory by posting messages recommended by its action part). If the actions recommended by different competing classifiers are contradictory, the classifier that has the largest *bid* has the greatest probability of being selected to fire. The bid of each activated classifier is proportional to its *fitness* or *strength* (Booker, Goldberg, & Holland, 1990).

Since many interdependent rules determine success or failure in solving the problem, a heuristically motivated *bucket-brigade* algorithm is used to change the strength of a classifier by apportioning credit or blame to it as a function of the environmental reward or punishment obtained at the end of a sequence of rule firings in which the classifier participated (Holland, 1975). Genetic operators are applied to a population of classifiers to generate new classifiers periodically which then participate in the bidding and credit assignment process.

Classifier systems offer an example of an effective combination of rule-based systems of the sort used in artificial intelligence in conjunction with numeric parameter modifying learning algorithms of the sort used in connectionist networks. Their simplicity and generality is appealing. However, it is neither always necessary nor always particularly desirable to restrict the representations used in such evolutionary systems to simple rules of the form used in classifier systems. We will return to this question later. But first, we examine genetic programming (Koza, 1992) which demonstrates the application of evolutionary learning methods to breed populations of `LISP` programs.

6.2 Representation and Learning in Genetic Programming Systems

Genetic programming (Koza, 1992) is an example of the application of evolutionary learning using populations of `LISP` programs. The use of syntactically correct `LISP` programs to represent potential solutions reduces the learning problem to the task of efficiently searching the space of `LISP` programs for an acceptable solution. Like Smith (1980), genetic programming circumvents the delicate problem of credit assignment to parts of a solution.

Given a set of primitive functions, an initial population of `LISP` functions is generated at random. The fitness of a program is evaluated, typically by executing the program on sample instances of the problem being learned. Thus, if the goal is to approximate an unknown function from examples, fitness evaluation can be based on some measure of error between the result of evaluating the `LISP` function and the desired result on a number of samples (argument values).

Genetic operators are applied to selected `LISP` functions from the current population to generate new functions. The mutation operator is generalized to operate on the parse tree (or nested list) representation of a `LISP` function. Thus mutation can randomly replace a primitive function appearing in the parse tree with another function of the same arity. A variety of other mutations are possible. In general, mutation does not alter the *shape* or the *structural complexity* of the function. Crossover on the other hand, results in exchange of sub-trees (functions, constants, or parameters) between the participating functions. The shape of the resulting trees (offspring) can in general be different from that of the parents.

Genetic programming has been applied to a variety of machine learning tasks (Koza, 1992) but its scalability to problems of realistic complexity remains to be established. However, it offers an intriguing approach to learning in domains where neither the size nor the shape of the solution is known a-priori. In this respect, it shares the philosophy behind generative algorithms developed in the context of connectionist networks.

6.3 Extending Evolutionary Learning

Classifier systems and the conventional genetic algorithms use extremely simple representations (typically fixed length strings) and simple genetic operators to explore large search spaces efficiently. The choice of representation and genetic operators ensures that the candidate solutions generated by the application of genetic operators are in fact legitimate members of the solution space for the problem. The representation based on bit-strings provide an effective and efficient basis for classification and abstraction whenever the bits in fact correspond to characteristic features. It is often possible to translate symbolic attribute-value based instance representation into bit-string representation and vice-versa (Booker, Riolo, & Holland, 1993). Abstract symbolic representations (e.g., semantic networks) can be translated into a system of classifiers (Forrest, 1991). None of this should be particularly surprising since rule-based production systems are Turing-equivalent (Post, 1943).

The fact that structured symbolic representations can be translated into classifier systems does not necessarily imply that learning has to be restricted to operate at the level of classifier systems. That would be tantamount to insisting that we do away with programming using high-level languages or abstract data types simply because they are eventually translated into machine language for execution on a digital computer.

Often, problems that a learner has to solve are posed in a language that provides building blocks that can be used to construct a solution. The learner should be able to exploit the descriptive power and the level of abstraction thus provided to solve such problem efficiently. It is worth exploring evolutionary learning techniques for a broad range of representations at varying levels of abstraction (e.g., those used in traditional artificial intelligence systems, computer vision, and syntactic pattern recognition (see above)). This essentially would entail the development of meaningful genetic operators for such representations. Such approaches offer an interesting range of alternatives between bit-string representations used in classifier systems and arbitrarily complex LISP programs.

As computing resources become less and less expensive, a number of interesting uses of the evolutionary paradigm suggest themselves. For example, genetic algorithms can work with structural specifications or blueprints of connectionist networks (e.g., types of neurons, functional modules, learning algorithms to be used, connectivity patterns, etc.). The fitness of individual networks can be measured in terms of how well they learn the problems posed to them by their environment. Genetic operations on the structural specifications of networks can be used to create new network architectures. Similarly, evolutionary algorithms may be used to select among different inductive and/or representational biases in families of learning programs according to the desired fitness criteria.

7 Summary

Learning structures and processes are essential components of adaptive, flexible, robust, and creative intelligent systems. Knowledge representation mechanisms play a central role in problem solving and learning. Indeed, learning can be thought of as the process of transforming observations or experience into knowledge to be stored in a form suitable for use whenever needed. Theoretical and empirical evidence emerging from investigations of learning within a number of research paradigms, using a variety of mathematical and computational tools strongly suggests the need for systems that can successfully exploit a panoply of representations and learning techniques.

It is unlikely that a single knowledge representation scheme or a single reasoning or knowledge transformation mechanism would serve all of the system's needs effectively in a complex environment. For example, answering questions about a road map is better handled given an iconic or picture-like representation with suitable set of operations to extract the necessary information than by using a bit-vector encoding that fails to make important spatial relations explicit. Similarly, a vision system capable of learning to recognize and describe complex 3-dimensional objects at different levels of detail must have at its disposal the representational structures to do so efficiently. When multiple tasks need to be learned and performed in a coordinated manner (e.g., in the case of a robot interacting in a complex environment) it is necessary for the representations to be seamlessly integrated in a structured fashion. The choice of the right representation(s) to use depends heavily on the task(s) to be performed and the inference mechanisms available.

We have examined some of the promising directions for synergistic integration of representations and learning techniques from multiple paradigms that are suggested by an examination of closely related but hithertofore largely disparate lines of research in artificial intelligence, connectionist networks, statistical and syntactic methods in pattern recognition, and evolutionary methods. Of these, traditional artificial intelligence and syntactic pattern recognition provide a number of powerful tools and techniques for manipulating predominantly symbolic structured representations while statistical pattern recognition and connectionist approaches provide the robustness that comes with the use of numeric weights and statistics. Evolutionary methods offer relatively efficient mechanisms for exploring large search spaces in the absence of other sources of information to direct the search. A careful examination of these paradigms suggests that there is really no fundamental incompatibility between them (Honavar & Uhr, 1993a; 1993b). The space of systems resulting from their systematic integration is full of potentially interesting possibilities. A priori it is difficult to determine exactly which subsets of these would be appropriate in a given system. To answer this question effectively, extensive empirical and (whenever feasible) theoretical studies are needed with a wide variety of such systems on a broad range of tasks.

8 References

Arbib, M. A. (1993). Schema Theory: Cooperative Computation for Brain Theory and Distributed AI. In this volume.

Ballard, D. H. (1987). Modular Learning in Neural Networks. In: *Proceedings of the Sixth National Conference on Artificial Intelligence.* Boston, Massachusetts: MIT Press.

Booker, L. E., Goldberg, D. E., & Holland, J. H. (1990). In: *Machine Learning Paradigms and Methods.* Carbonell, J. (Ed). Boston, Massachusetts: MIT Press.

Booker, L. E., Riolo, R. L., & Holland, J. H. (1993). Learning and Representation in Classifier Systems. In this volume.

Buchanan, B. G., & Wilkins, D. C. (1993). Readings in Knowledge Acquisition and Learning. San Mateo, California: Morgan Kaufmann.

Carpenter, G., & Grossberg, S. (1991). (Ed). Pattern Recognition by Self-Organizing Neural Networks. Cambridge, Massachusetts: MIT Press.

Devijver, P. & Kittler, J. (1982). Pattern Recognition: A Statistical Approach. Englewood Cliffs, New Jersey: Prentice-Hall.

Duda, R. O., & Hart, P. E. (1973). Pattern Classification and Scene Analysis. New York: Wiley.

Fahlman, S. E., & Lebiere, C. (1990). The Cascade-Correlation Architecture. In: *Advances in Neural Information Processing Systems Vol. 2.* Touretzky, D. S. (Ed). San Mateo, CA: Morgan Kaufmann.

Fodor, J. (1976). The Language of Thought. Boston, MA: Harvard University Press.

Fogel, L. J., Owens, A. J., & Walsh, M. J. (1966). Artificial Intelligence Through Simulated Evolution. New York: Wiley.

Fu, K. S. (1982). Syntactic Pattern Recognition and Applications. Englewood Cliffs, New Jersey: Prentice-Hall.

Fukunaga, K. (1990). Introduction to Statistical Pattern Recognition. New York: Academic Press.

Gallant, S. (1993). Neural Network Learning and Expert Systems. Cambridge, Massachusetts: MIT Press.

Gennari J., Langley, P., & Fisher, D. (1990). Models of Incremental Concept Formation. In: *Machine Learning Paradigms and Methods.* Carbonell, J. (Ed). Boston, Massachusetts: MIT Press.

Gentner, D. (1993). The Mechanisms of Analogical Learning. In: *Readings in Knowledge Acquisition and Learning.* Buchanan, B. G., & Wilkins, D. C. (Ed). San Mateo, California: Morgan Kaufmann.

Ginsberg, M. (1993). Essentials of Artificial Intelligence. San Mateo, California: Morgan Kaufmann.

Goldberg, D. E. (1989). Genetic Algorithms in Search, Optimization, and Machine Learning. New York: Addison Wesley.

Goldfarb, L. (1990). An Evolving Model for Pattern Learning. *Pattern Recognition 23*, 595-616.

Gonzalez, R. C., & Thomason, M. G. (1978). Syntactic Pattern Recognition: An Introduction. Reading, Massachusetts: Addison Wesley.

Hendler, J. (1989). The design and Implementation of Marker-Passing Systems. *Connection Science* 1 17-40.

Hinton, G. E. (1990). Connectionist Learning Procedures. In: *Machine Learning Paradigms and Methods.* Carbonell, J. (Ed). Boston, Massachusetts: MIT Press.

Holland, J. H. (1975). Adaptation in Natural and Artificial Systems. Ann Arbor, Michigan: University of Michigan Press.

Honavar, V. & Uhr, L. (1988). A Network of Neuron-Like Units that Learns to Perceive by Generation as well as Reweighting of its Links. *Proceedings of 1988 Connectionist Models Summer School*, Touretzky, D., Hinton, G and Sejnowski T. (Eds.), San Mateo, California: Morgan Kaufmann.

Honavar, V. & Uhr, L. (1989a). Brain-Structured Connectionist Networks that Perceive and Learn, *Connection Science 1*, pp. 139-159.

Honavar, V. & Uhr, L. (1989b). Generation, Local Receptive Fields, and Global Convergence Improve Perceptual Learning in Connectionist Networks. In: *Proceedings of the 1989 International Joint Conference on Artificial Intelligence*, San Mateo, California: Morgan Kaufmann.

Honavar, V. & Uhr, L. (1990). Coordination and Control Structures and Processes: Possibilities for Connectionist Networks. *Journal of Experimental and Theoretical Artificial Intelligence* 2 277-302.

Honavar, V. (1992a). Inductive Learning Using Generalized Distance Measures. In: *Proceedings of the SPIE Conference on Adaptive and Learning Systems.* Orlando, Florida.

Honavar, V. (1992b). Some Biases for Efficient Learning of Spatial, Temporal, and Spatio-Temporal Patterns. In: *Proceedings of the International Joint Conference on Neural Networks.* Beijing, China.

Honavar, V. (1992c). Learning Parsimonious epresentations of Three-Dimensional Shapes. In: *NATO Advanced Research Workshop on Mathematical Representations of Shape*, Driebergen, Netherlands. To appear.

Honavar, V. & Uhr, L. (1993) Generative Learning Structures and Processes for Generalized Connectionist Networks. *Information Sciences* (Special Issue on Artificial Intelligence and Neural Networks) 70 75-108.

Honavar, V., & Uhr, L. (1993a). Symbol Processing Systems, Connectionist Networks, and Beyond. In: *Intelligent Hybrid Systems.* Goonatilake, S. & Khebbal, S. (Ed). London: Wiley. To appear.

Honavar, V., & Uhr, L. (1993b). Toward Integration of Symbol Processing and Connectionist Systems in Artificial Intelligence and Cognitive Modelling. In: *Computational*

Architectures for Integrating Symbolic and Neural Processes. Sun, R., & Bookman, L. (Ed). New York: Kluwer. To appear.

Jacobs, R. A., Jordan, M. I., & Barto, A. G. Task Decomposition Through Competition in a Modular Connectionist Architecture: The What and Where Vision Tasks. Tech. Rep. 90-27. Amherst, Massachusetts: Department of Computer and Information Science, University of Massachusetts.

Knuth, D. E. (1968). Semantics of Context-Free Languages. *Journal of Mathematical System Theory* 2 127-146.

Kohonen, T. (1989). Self-Organization and Associative Memory. Berlin: Springer-Verlag.

Koza, J. (1992). Genetic Programming. Boston, Massachusetts: MIT Press.

Kung, S. Y. (1993). Digital Neural Networks. New York: Prentice Hall.

Laird, J. E., Rosenbloom, P. S. & Newell (1986). Chunking in SOAR: The Anatomy of a General Learning Mechanism. *Machine Learning 1*, 11-46.

Langley, P., & Zytkow, J. M. (1990). Data-Driven Approaches to Empirical Discovery. In: *Machine Learning Paradigms and Methods.* Carbonell, J. (Ed). Boston, Massachusetts: MIT Press.

Lenat, D. B. (1982). AM: Discovery as Heuristic Search. In: *Knowledge-based Systems* Davis, R., & Lenat, D. B. (Ed). New York: McGraw-Hill.

Levenshtein, A. (1966). Binary Codes Capable of Correcting Deletions, Insertions, and Reversals. *Soviet Physics - Doklady 10*, 703-710.

Michalewicz, Z. (1992). Genetic Algorithms + Data Structures = Evolution Programs. New York: Springer-Verlag.

Michalski, R. S. (1983). Theory and Methodology of Inductive Learning. In: *Machine Learning – An Artificial Intelligence Approach.* Michalski, R. S., Carbonell, J. G., & Mitchell, T. M. (Ed). Palo Alto, California: Tioga.

Michalski, R. S. (1993). Toward a Unified Theory of Learning: Multi-Strategy Task-Adaptive Learning. In: *Readings in Knowledge Acquisition and Learning.* Buchanan, B. G., & Wilkins, D. C. San Mateo, California: Morgan Kaufmann.

Miclet, L. (1986). Structural Methods in Pattern Recognition. New York: Springer-Verlag.

Minsky, M. (1975). A Framework for Representing Knowledge. In: *The Psychology of Computer Vision.* Winston, P. H. (Ed). New York: McGraw-Hill.

Minton, S., Carbonell, J. G., Knoblock, C. A., Kuokka, D. R., & Gil, Y. (1990) In: *Machine Learning Paradigms and Methods.* Carbonell, J. (Ed). Boston, Massachusetts: MIT Press.

Mitchell, T. (1982). Generalization as Search. *Artificial Intelligence* 18 203-226.

Natarajan, B. K. (1992). Machine Learning: A Theoretical Approach. San Mateo, California: Morgan Kaufmann.

Newell, A., & Simon, H. A. (1972). Human Problem-Solving. Englewood Cliffs, New Jersey: Prentice-Hall.

Newell, A. (1980). Symbol Systems. *Cognitive Science* 4 135-183.

Omlin, C. W., & Giles, C. L. (1993). Extraction and Insertion of Symbolic Information in Recurrent Neural Networks. In this volume.

Parekh, R. G., & Honavar, V. (1993a). Efficient Learning of Regular Languages Using Teacher-Supplied Positive Samples and Learner-Generated Queries. In: *Proceedings of the UNB Artificial Intelligence Symposium.* Fredericton, Canada. To appear.

Parekh, R. G., & Honavar, V. (1993b). Grammar Induction for Machine Learning. In preparation.

Pavlidis, T. (1977). Structural Pattern Recognition. New York: Springer.

Post, E. (1943). Formal Reduction of the General Combinatorial Decision Problem. *American Journal of Mathematics* 65 197-268.

Rosenfeld, A. (1981). Picture Languages. New York: Academic Press.

Rosenfeld, A. (1984). (Ed). Multi-Resolution Image Processing and Analysis. New York: Springer-Verlag.

Rueckl, J. G., Cave, K. R., & Kosslyn, S. M. (1989). Why Are "What" and "Where" Processed by Separate Visual Systems? A Computational Investigation. *Journal of Cognitive Neuroscience* 1 171-186.

Rumelhart, D. E., & McClelland, J. L. (1986) (Ed). *Parallel Distributed Processing, Vol. I-II*, Cambridge, Massachusetts: MIT Press.

Shavlik, J. W., & Dietterich, T. G. (1990). (Ed). Readings in Machine Learning. San Mateo, California: Morgan Kaufmann.

Shavlik, J. W. (1993). A Framework for Combining Symbolic and Neural Learning. In this volume.

Shrager, J., & Langley, P. (1990). Computational Models of Scientific Discovery and Theory Formation. San Mateo, CA: Morgan Kaufmann.

Simon, H. A. (1983). Why Should Machines Learn? In: *Machine Learning – An Artificial Intelligence Approach.* Michalski, R. S., Carbonell, J. G., & Mitchell, T. M. (Ed). Palo Alto, California: Tioga.

Sowa, J. F. (1984). Conceptual Structures: Information Processing in Mind and Machine. Reading, Massachusetts: Addison-Wesley.

Smith, S. F. (1980). A Learning System Based on Genetic Operators. Ph.D. Dissertation. University of Pittsburgh.

Smolensky, P. (1990). Tensor Product Variable Binding and the Representation of Symbolic Structures in Connectionist Systems. *Artificial Intelligence* 46 159-216.

Sun, R. (1993). Logic and Variables in Connectionist Models: A Brief Overview. In this volume.

Tanimoto, S. L., & Klinger, A. (1980). Structured Computer Vision: Machine Perception Through Hierarchical Computation Structures. New York: Academic Press.

Tesauro, G. (1989). Neurogammon Wins Computer Olympiad. *Neural Computation* 1 321-323.

Uhr, L., & Vossler, C. (1963). A Pattern Recognition Program that Generates, Evaluates, and Adjusts its Own Pperators. In: *Computers and Thought.* Feigenbaum, E., & Feldman, J. (Ed). New York: McGraw-Hill.

Uhr, L. (1973). Pattern Recognition, Learning, and Thought. New York: Prentice-Hall.

Uhr, L. (1987) (Ed). Parallel Computer Vision. New York: Academic Press.

Valiant, L. G. (1984). A Theory of the Learnable. *Communications of the ACM* 27 1134-1142.

Waibel, A. (1989). Modular Construction of Time-Delay Neural Networks for Speech Recognition. *Neural Computation* 1 39-46.

Waterman, D. A. (1985). A Guide to Expert Systems. Reading, Massachusetts: Addison-Wesley.

Winston, P. H. (1975) (Ed). The Psychology of Computer Vision. New York: McGraw-Hill.

Winston, P. H. (1992). Artificial Intelligence. Boston, Massachusetts: Addison-Wesley.

Yang, J. & Honavar, V. (1993). A Hierarchical Representation Scheme for Three-Dimensional Object Recognition and Description. In: *Proceedings of FLAIRS-93*, Fort Lauderdale, Florida.

Tech Report: TR93-22
Submission Date: September 14, 1993